

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

国内超大规模Zabbix集群负责人力作

全面讲解Zabbix配置应用，深入剖析Zabbix内部原理

用真实工作需求驱动，以独家实践案例指引，助您监控利器出鞘

Broadview[®]
www.broadview.com.cn




Zabbix

监控系统深度实践

第2版

姚仁捷◎著

 中国工信出版集团

 电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

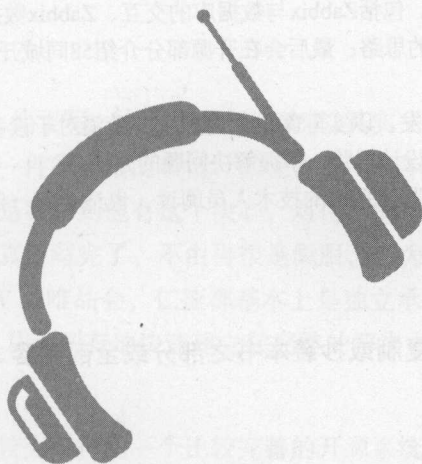


◀ 姚仁捷

运维数据与开发经理，现就职于游族网络（002174.SZ），负责运维与大数据方面的开发，包括日志平台、计算平台和机器学习方面的研究与开发。

国内最早研究Zabbix的技术人员之一，Zabbix Python API作者。曾在PPTV负责当时Zabbix社区中最大的集群之一。对Zabbix大规模集群有丰富的经验，善于Zabbix源码的改造和开发。作为Reviewer参与了PACKT Publishing出版的*Zabbix Performance Tuning*审阅工作。

目前对于机器学习有浓厚的兴趣，希望能将大数据、机器学习和运维结合，使得数据化运维能够真正落地。



Zabbix

监控系统深度实践

第2版

姚仁捷◎著

电子工业出版社

Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书由浅入深,全面讲解Zabbix应用与原理,是作者多年实战经验的总结和浓缩。在概念篇,从一个简单但完整的入门案例讲起,案例中有最基本的概念介绍,通过案例帮助那些只需将服务器加入监控,就能看到监控数据的读者;然后逐步深入,在进阶篇介绍Zabbix的各方面的配置;在设计篇中对Zabbix的内部原理进行深入剖析,包括Zabbix与数据库的交互、Zabbix数据库表的设计等,并分享作者在Zabbix上踩过的坑以及解决问题的思路;最后会在开源部分介绍58同城开源的Zatree和Chrome的插件、手机客户端等工具。

本书从工作中的实际需求出发,以实际案例作为指引,希望对于读者而言,不仅仅是学会某些具体的操作,而是深入了解Zabbix的设计思路,掌握解决问题的方法。

本书适合想使用Zabbix构建监控系统的技术人员阅读,也适合有一定基础、对于Zabbix有更高的要求读者。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

Zabbix监控系统深度实践/姚仁捷著. —2版. —北京:电子工业出版社,2016.8
ISBN 978-7-121-29608-6

I. ①Z… II. ①姚… III. ①计算机监控系统 IV. ①TP277

中国版本图书馆CIP数据核字(2016)第181578号

责任编辑:董 英

印 刷:北京嘉恒彩色印刷有限责任公司

装 订:北京嘉恒彩色印刷有限责任公司

出版发行:电子工业出版社

北京市海淀区万寿路173信箱 邮编:100036

开 本:787×980 1/16 印张:23.5 字数:527千字

版 次:2014年8月第1版

2016年8月第2版

印 次:2017年5月第2次印刷

印 数:3001~4000册 定价:79.00元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010)88254888,88258888。

质量投诉请发邮件至zlts@phei.com.cn,盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式:(010)51260888-819,faq@phei.com.cn。

姚仁捷同学跟我提起,他要写一本关于 Zabbix 的书,其实一开始我是不太鼓励的。在我看来,写书向大众传播知识,是一件很严肃的事情,仁捷作为一名年轻的技术人员,去完成一本书可能还是会有些吃力的。但是我看到他有这个决心,对待书的态度也非常虔诚,每天都会在繁忙的工作之余加班写作,书真的写完了,不由得很是佩服,作为他毕业到现在的多年老板,必须顶一下。事实上,从 PPTV 到唯品会,仁捷都基本上是独立承担一个领域的工作,借鉴业界的最佳实践 (Best Practice),从无到有地快速建立起完整的解决方案。的确是,聪明的人,给机会,就能脱颖而出。

Zabbix 是业界近年来较为流行的一个比较完善的开源系统监控解决方案,我们当初也是调研了不少解决方案才选择了它。姚仁捷曾经是 PPTV 的 Zabbix 集群的负责人。PPTV 的 Zabbix 应用,已经是国内较大规模的系统监控了,覆盖了当时 5000 多台主机和上面应用的几十万个监控点,近百万的监控点记录,也修改了 Zabbix 多处源码,实现了很多自动化的监控部署和 Proxy-Master 的分布式监控,以及通过 Zabbix Trigger 自动分析等,也就 Zabbix 本身的一些缺点设计了对应的 workaround 的办法和二次开发,可以说是国内较为领先的大规模部署解决实际案例。在 Zabbix 的实践领域,PPTV 的很多方法、思想和技巧都很有价值,仁捷同学在这本书中也都有涉及。

好的经验还是值得分享的,就算还不是完美的。

希望这本书能够给大家带来一手的 Zabbix 实战经验,更加希望大家可以从中借鉴作者分享的经验,少走弯路,帮助公司更加多快好省地建设系统监控解决方案。

唯品会高级总监

诸超

Zabbix 作为一款企业级的、开源的、分布式的监控套件，设计理念超前，解决了以往监控软件的短板，可以说是现在最流行的监控解决方案之一。

Zabbix 可以监控网络和服务的健康状况，可以利用模板批量添加服务器，可以自定义监控项，可以利用灵活的报警机制给运维人员发送 E-mail 和短信报警，从而保证了运维人员能快速对问题作出响应。此外，Zabbix 简单易上手，只要稍作学习，就能迅速搭建一套运维监控平台，瞬间高大上。

Zabbix 在分布式方面做了大量的优化工作，这样可以保证在多机房和对海量服务器进行监控时，能快速高效地收集数据，并集中在一个界面内展示。不过目前我所负责项目中，15 万个 Items 和 1000 多个 Hosts 用了一个配置比较高的服务器在抗，毫无压力，等服务器规模再大一些，机房比较多的时候，我会考虑用分布式。

我使用 Zabbix 也快 1 年了，替换了原来的 Nagios+Cacti 方案。Zabbix 兼有 Nagios+Cacti 的特点，所以现在维护一个系统就可以了，极大地方便了运维工作。因为公司大部分都是标准化的服务和服务器，迁移过程也比较顺利，只要事先做好分组，设置几个模板一关联就可以了，迁移的大部分时间花在了寻找合理阈值和设置靠谱 Trigger 上，这个可能需要慢慢积累经验。

作为 Zabbix 插件 Zatree 的开发者之一，我一直比较关注 Zabbix 在国内的发展，这几年来是 Zabbix 发展的快速时期，大量爱好者在 QQ 群、微博和社区参与讨论和分享，极大地丰富了 Zabbix 的中文资料。作为曾经国内最大规模 Zabbix 集群的负责人，姚仁捷在本书中全面讲解了 Zabbix 的安装、配置、使用及技巧，提供了大量的案例和解决问题的心得，其中也介绍了 Zatec 插件的安装和使用，希望大家可以一边看书，一边亲自动手实践，这样效果会更好。

相信人人都能成为监控专家、运维专家。

中国最大开源社区 Chinaunix 创始人之一

窦喆

@ 南非蜘蛛

前言

本书的由来

我从职业生涯开始至今，就一直在和监控系统打交道。

我最早在 eBay 容量规划小组工作，使用监控系统查看服务器状态及网站运营指标；后来到了 PPTV 运维部，通过监控系统的数据了解上线发布的结果和网站的健康程度等情况；现在到了唯品会，我们的监控系统能够从业务、技术两个维度考察当前公司网站的运作情况。

在有监控系统之前，工程师需要到服务器上去敲命令来获取系统数据；为了分析问题，可能还需要将数据复制到本地计算机的 Excel 里进行画图；最要命的是，在出现问题的时候无法知道，只有在用户报障后才能察觉。这是多么骇人听闻的场景！

而当我们有了一个好的监控系统后，这些问题就迎刃而解了。我们可以在一个界面中浏览整个机房的服务器状态、可以在 Web 前端方便地查看监控数据、可以回溯寻找事故发生时系统的问题和报警情况。现在，我们的工程师们已经可以一边悠闲地喝着咖啡一边分析问题了。

监控系统是整个运维自动化体系中非常重要的环节。从服务器上架到最后被回收重用，都有监控系统的身影。服务器上架时，它需要添加监控；在服务器工作过程中，监控系统要时刻注意服务器的健康，并且在服务器出现异常时，要发出报警通知对应的人员；在服务器被回收时，监控系统要取消服务器的监控。这些都需要监控系统拥有 API，能够方便地跟外部其他系统一起工作，把自己的工作自动化起来。

国内的互联网巨头们，可以自行开发一套监控系统。而对于绝大多数企业来说，开源的 Zabbix 是非常棒的选择。它能够非常好地实现以上这些需求。可以说，目前 Zabbix 是最热门的开源监控系统。

本书的内容结构

从周围的 QQ 群、论坛等地方，我发现大家对于 Zabbix 的学习都是非常零散的，缺少一个系统的学习过程和解决问题的正确思路。在这本书的前面，我会先向大家介绍一个最简单的入门案例，案例中有最基本的概念介绍，通过案例帮助那些只需将服务器加入监控，并且看到监

控数据的读者。后面深入一些，会介绍 Zabbix 的方方面面的配置，适合打算使用 Zabbix 高级功能的读者。在接下来的部分，会深入剖析 Zabbix 的内部原理，包括 Zabbix 与数据库的交互、Zabbix 数据库表的设计等我在 Zabbix 上踩过的坑以及解决问题的思路。希望能授之以渔。在本书的最后部分，主要介绍 Zabbix 在开源方面的进展，最主要的就是 58 同城开源的 Zatree，以及 Chrome 的插件和手机客户端。

本书会从我们工作中的实际需求出发，介绍 Zabbix 的使用方法和其配置管理。在这些内容之后，会有深入一些的对于 Zabbix 实现的讲解，希望对于读者而言，不仅仅是学会某些具体的操作，而是深入了解 Zabbix 的设计思路，掌握解决问题的方法。

作者联系方式

由于经验的不足，书中可能会有些不足之处，大家可以通过微博 @ 超大杯摩卡星冰乐，或者邮箱 baniu.yao@gmail.com，与我联系。

声明

在刚开始进行写作时，我考虑到很多读者是用中文版的 Zabbix，所以文中的 Zabbix 的术语都使用中文。但后来我觉得对于 Zabbix 的术语，研究人员是需要了解它的英文说法的，这样在同行之间才能更好地交流，也可以在 Google 上更好地检索信息。基于这个原因，我将之前的中文术语全部又换成了英文。由于这些术语非常多，虽然编辑帮我细致地进行了检查，难免有疏漏，希望大家能够谅解。

致谢

在前言的最后，要感谢很多人。首先感谢的是我的父母，没有你们，就没有我。然后要感谢我的老婆，因为要忙于写书，很多时候不能陪你。最后要感谢的是诸超、陈文春、吴晓刚、周昕毅、朱宁和刘海阳等同事的帮助，在我写书的过程中，给出了很多宝贵的建议。谢谢各位。

目录

第一部分 概念篇

第 1 章 自动化运维和监控系统	2
1.1 互联网公司的运维工作	2
1.2 何谓自动化运维	3
1.3 监控系统在运维自动化中的角色	5
1.4 监控系统的理想化模样	5
第 2 章 Zabbix简介	7
2.1 Zabbix发展现状	7
2.2 选择Zabbix的理由	8
2.3 Zabbix部分名词约定	9
第 3 章 Zabbix安装	11
3.1 获取Zabbix	11
3.2 Zabbix Server安装	12
3.2.1 Zabbix数据库配置	12
3.2.2 安装Zabbix Server	13
3.2.3 安装Zabbix Web前端	16
3.3 Zabbix Agent安装	18
3.3.1 UNIX/Linux上安装Zabbix Agent	18
3.3.2 Windows上安装Zabbix Agent	18
3.4 测试Zabbix Agent和Zabbix Server运行	20
3.5 配置文件详解	20
3.5.1 zabbix_server.conf	20
3.5.2 zabbix_agentd.conf	24

第4章 监控第一台Host	26
4.1 Host在监控系统中的活动	26
4.2 添加一个用户	27
4.3 把服务器加入Zabbix监控	27
4.4 添加Item	28
4.5 添加Trigger	29
4.6 设置Action	31
4.7 收到第一封报警邮件	33
4.8 Zabbix 报警流程	33
4.9 看, Zabbix在工作呢	34
4.9.1 全局搜索框	35
4.9.2 查看监控数据	35
4.9.3 查看报警信息	36
4.10 添加自定义监控点	37

第二部分 配置篇

第5章 增加监控	40
5.1 Host配置	41
5.2 Item属性	45
5.3 Item类型	48
5.3.1 Zabbix Agent类型	48
5.3.2 SNMP类型	51
5.3.3 IPMI类型	52
5.3.4 日志文件监控	53
5.3.5 计算型Item	54
5.3.6 Zabbix内部监控	55
5.3.7 ssh类型Item	58
5.3.8 Telnet类型Item	60
5.3.9 External Check类型Item	60

5.3.10	Aggregate类型Item	60
5.3.11	Trapper类型Item	62
5.3.12	JMX类型Item	62
5.3.13	ODBC类型Item	64
5.4	Item历史数据History和Trends	66
5.5	使用Application对Item分组	67
5.6	Item Key详解	68
5.7	Template模板	69
5.7.1	新建和配置一个Template	69
5.7.2	建立/取消Host和Template的关联	71
5.7.3	修改Template	73
5.7.4	Template和Host	73
5.7.5	Template之间的父子关系	74
5.8	Clone、Full Clone和Mass Update	75
5.9	Windows监控	76
5.10	VMware监控	82
5.11	Zabbix监控性能	84
第6章	报警配置	86
6.1	Triggers	86
6.1.1	配置Triggers	86
6.1.2	Trigger expression	87
6.1.3	Function详解	89
6.1.4	Trigger依赖	92
6.1.5	Trigger等级	94
6.1.6	单位	95
6.2	Events	95
6.3	Actions	96
6.3.1	Action	97
6.3.2	Operation	99

6.3.3	Condition	104
6.3.4	Escalations	107
6.3.5	Unsupported状态的Items的报警	110
6.4	Media类型	111
6.5	Maintenance状态	116
第 7 章	数据可视化	118
7.1	Graph	118
7.2	Network Maps	123
7.2.1	新建Maps	123
7.2.2	创建元素	124
7.2.3	选择元素	126
7.2.4	关联元素	126
7.2.5	关联指示器	126
7.3	Screens	127
7.4	Slide shows	131
第 8 章	Users和Macros	133
8.1	User和用户group	133
8.1.1	配置User	133
8.1.2	User group	135
8.2	Macros	136
8.2.1	自带宏	136
8.2.2	用户自定义宏	137
8.2.3	自定义宏的适用范围	139
第 9 章	IT services服务监控与Web monitoring网络监控	140
9.1	Services服务监控	140
9.2	服务配置	141
9.3	Web monitoring网络监控配置	145
9.4	监控百度示例	148

第 10 章 Zabbix前端界面	151
10.1 Monitoring板块	151
10.1.1 Dashboard栏目	151
10.1.2 Overview栏目	157
10.1.3 Web栏目	158
10.1.4 Latest data栏目	159
10.1.5 Triggers栏目	159
10.1.6 Events栏目	160
10.1.7 Graphs&Screens&Maps栏目	161
10.2 Inventory板块	161
10.3 Reports板块	161
10.4 Configuration板块	166
10.4.1 Host groups栏目	166
10.4.2 Template栏目	167
10.4.3 Hosts栏目	168
10.4.4 Maintenance栏目	170
10.4.5 其他	170
10.5 Administration板块	171
10.5.1 General栏目	171
10.5.2 DM栏目	177
10.5.3 Authentication栏目	178
10.5.4 Users栏目	179
10.5.5 Media types栏目	181
10.5.6 Scripts栏目	181
10.5.7 Audit栏目	185
10.5.8 Queue栏目	186
10.5.9 Notification栏目	186
10.5.10 Installation栏目	187
10.6 前端配置	187

10.6.1	全局配置参数	187
10.6.2	前端维护状态显示	189
10.6.3	Profile设置	190
10.7	全局搜索框	192
第 11 章	Discovery	193
11.1	基于网络的Discovery	193
11.2	Discovery的一个例子	195
11.3	Discovery Rule和Discovery Action的配置	196
11.4	存活Agent自动加入监控	199
11.5	low-level discovery	200

第三部分 进阶篇

第 12 章	Zabbix API	206
12.1	Zabbix API POST参数	206
12.2	Item支持的Zabbix API方法	207
12.2.1	Item object	208
12.2.2	item.create	209
12.2.3	item.delete	210
12.2.4	item.exists	210
12.2.5	item.get	211
12.2.6	item.getobjects	214
12.2.7	item.isreadable/item.iswritable	215
12.2.8	item.update	215
12.3	如何阅读Zabbix API文档	216
第 13 章	Zabbix分布式监控	217
13.1	两种分布式架构对比	217
13.2	Proxy单级分布式架构	218
13.3	Proxy配置	219

13.4 Node多级分布式架构	220
第 14 章 Zabbix系统优化	227
14.1 Zabbix内部运行机制	227
14.2 Items过多造成性能下降	228
14.3 数据库及其他调优	232
第 15 章 轻量级日志监控应用	233
15.1 准备工作	233
15.2 添加 Item	234
15.3 测试	234
15.4 配置报警	236
15.5 轮转的日志文件	237
15.6 获取关键字	238

第四部分 设计篇

第 16 章 Zabbix数据库表结构解析	240
16.1 表结构概述	240
16.2 Hosts表	241
16.3 Items表	244
16.4 Trigger在数据库中的结构	248
16.5 Events表	253
16.6 Triggers和Events生成的规则	255
第 17 章 History和Trends	256
17.1 sync字段的含义	257
17.2 history和trends的区别	261
17.3 housekeeper和trends表	262
17.4 Graph对于history和trends的选择	263
第 18 章 Zabbix和数据库交互详解	268
18.1 include/zbxdb.h	268

18.2	zbxdb/db.c	270
18.3	zbxdbhigh	271
第 19 章 Zabbix 2.2新功能介绍		274
19.1	数据库自动升级	274
19.1.1	检查数据库版本	274
19.1.2	mandatory和optional字段	275
19.1.3	数据库升级过程	277
19.1.4	前端提示	278
19.2	Web监控	279
19.2.1	Web监控Template化	279
19.2.2	Web监控重试机制	279
19.2.3	使用HTTP代理	280
19.2.4	URL监控中使用页面内容作为变量	281
19.3	数据映射	282
19.4	history和trends存储的代码分析	282
19.4.1	DCsync_history	283
19.4.2	DCsync_trends	285
19.4.3	整个流程	285
19.5	网页字符串匹配	286
19.6	日志文件监控	287
19.7	Latest Data局部刷新	288
19.8	动态载入模块	288
19.9	SNMP监控改进	292
19.9.1	SNMPv3相关的增强	292
19.9.2	SNMP重试和超时机制改进	293
19.9.3	lld的复杂OIDs	293
第 20 章 Zabbix内置监控项实现		294
20.1	system.hostname	294
20.2	system.cpu.load	295

第五部分 社区和开源

第 21 章 典型案例分析	300
21.1 前端显示Zabbix server停止工作问题	300
21.2 Item设置了但没有数据	306
21.2.1 看页面是否有报错	306
21.2.2 Zabbix Server和Zabbix Agent的网络是否互通	307
21.2.3 zabbix_get是否能够获取到数据	308
21.2.4 总结	308
21.3 一个扫描history全表的SQL问题	309
21.4 解决问题的思路	319
第 22 章 Zabbix代码问题和解决	320
22.1 Duplicated Host问题	320
22.2 拼接大SQL问题	322
22.3 nextid问题	323
22.4 在Zabbix中打印日志	325
第 23 章 PPTV的Zabbix监控体系	326
23.1 Python Zabbix API	326
23.2 Spider——服务器添加Zabbix监控	328
23.3 Event Console	330
23.4 Rule Engine	330
23.5 报警系统架构	331
第 24 章 Zatree	332
24.1 使用Zatree	332
24.2 Zabbix二次开发和重新开发监控系统的选择	334
第 25 章 Zabbix第三方插件	337
25.1 Chromix	337
25.2 Zabbix Notifier	338

25.3 手机端Zabbix App	339
25.3.1 ZBX Mobile	339
25.3.2 Zabbix	341
第 26 章 微信公众平台报警	344
26.1 申请微信公众平台账号	344
26.2 配置微信公众平台账号	345
26.2.1 使用SAE进行测试开发	347
26.2.2 申请测试账号	348
26.2.3 获取access_token	348
26.2.4 获取用户的openid	349
26.2.5 发送第一条文字消息	349
26.3 微信接口请求次数限制	350
第 27 章 社区论坛	351
附录 Zabbix自带宏	353
后记	355
程序员职业生涯的一些感悟	356

章上第

从这几条故障，基本能理解运维人员的痛苦了吧。下面说说我在工作过程中遇到的最烦闷事情的痛苦：

- (1) 系统出问题除了不知道，要搞到客户投诉，然后针对服务出的问题。
- (2) 发布信息太痛苦了。要一台一台机器去执行命令，一台一台机器去检查发布是否成功。
- (3) 故障之后不能。
- (4) 分析问题的根。
- (5) 不能是资产情况。很多公司的运维人员，连不知道自己公司的资产有多少台服务器。

1.2 何谓自动化运维

自动化运维和监控系统



第一部分 概念篇

★ 第1章 自动化运维和监控系统

★ 第2章 Zabbix 简介

★ 第3章 Zabbix 安装

★ 第4章 监控第一台 Host

第 1 章

自动化运维和监控系统

1.1 互联网公司的运维工作

一般来说，公司中的运维，包括基础运维、应用运维、运维开发和监控组四部分。前两者是必需的——基础运维是负责 IDC 的，上架服务器、网络这些；应用运维可以理解为 SA（System Administrator），即系统管理员；而运维开发在一些公司则不属于运维部门，主要负责运维工具开发、系统开发等工作，比如监控系统、发布系统；监控组，就是 24 小时值班的人员，他们要时刻关注网站或者系统的状况，一旦出现问题，要第一时间进行排查，并且联系相关的运维和研发工程师。

国内中大型的互联公司，拥有几百、几千台服务器是很正常的情况，所以运维工程师一直是非常“悲催”的职业。每天在几千台服务器上面敲命令，查看系统状态，发布代码，任务很是繁重。

在微博上，搜索“运维苦逼”关键字，可以看到很多不堪重负的疲惫语言，比如下面几条：

（1）双 11 来了，今晚支付宝都瘫痪啦，我在想今晚程序员，搞运维的，项目团队又要苦逼地加班了。

（2）atlassian 家的东西怎么和我家教务一样呢……这一天都挂成狗了，Java 运维好苦逼。

（3）双 11 最苦逼的是谁？看着老婆抢购的汉子？No！是淘宝的运维，他和他男朋友们一起在通宵加班啊。

.....

从这几条微博,基本能理解运维人员的痛苦了吧。下面说说我在工作过程中碰到的运维同事们的痛苦:

- (1) 系统出问题了不知道。要等到客户报障才知道网站服务出问题。
- (2) 发布代码太痛苦了。要一台一台机器去执行命令,一台一台机器去检查发布是否成功。
- (3) 机器之间不统一。代码是一样的,一台机器是正常的,另一台是不正常的,见鬼了!
- (4) 分析问题困难。比如想看过去1天的CPU负载的情况,必须先用 `sar -q` 获取到这些数据,然后复制到 Excel,再画图。如果想几台机器一起看,那更加痛苦。
- (5) 不清楚资产情况。很多公司的运维人员,竟然不知道自己公司的 IDC 有多少台服务器。

1.2 何谓自动化运维

如果读者是运维工程师,看到前一小节结尾的时候,肯定嘴角露出邪恶的笑容,心里想“这我全部碰到过了”。是啊,这些是苦逼运维工程师几乎每天要面对的问题。反过来想,以这些“痛苦”作为反例,我们能想到一个好的运维架构是怎样的:

- (1) 硬件标准化——包括服务器、内存、系统版本等。
- (2) 软件标准化——应用版本等。
- (3) 运维自动化——包括监控、发布、CMDB。

前两点都很容易理解,那运维自动化具体是干什么的呢?

笔者的理解,运维自动化,就是把运维中大量日常重复性工作使用工具让其自动运行,减少人的参与。首先它要包括以下几部分:

- (1) 监控报警——系统数据,应用指标的监控,和出错时及时报警。
- (2) 发布系统——代码发布,发布后的检查,代码的回滚,灰度发布。
- (3) 服务器标准化——Cobbler 装机加上 Puppet,可以做到硬件、软件的标准化。每台机器对于运维来说都是一样的。
- (4) CMDB——配置管理数据库,存储了所有运维相关数据,包括服务器硬件信息、域名和服务器的关系、IDC 容量等。它是运维的心脏,所有的系统都依赖于它。

说白了，这就是运维自动化了，前面提到的痛苦全都“药到病除”了：系统出问题了？报警邮件马上就发给你，简单的错误甚至可以自动修复，使监控数据的可视化、性能分析变得轻松；代码发布困难？使用发布系统，代码首先灰度发布，发布一些机器检查正常后，继续发布（在PPTV已经能做到一键发布，一键回滚）；机器不统一？Cobbler保证装出来的服务器完全一样，Puppet保证服务器上的软件都是一个版本；资产容量搞不清？CMDB帮你理清楚一切。

当公司只有几十台服务器的时候，“人肉”进行这些工作去解决不标准的问题，都是可以的，无非是多找一些工程师罢了。但当公司的规模扩大，服务器到了几百、几千的规模后，还能这样做吗？答案是不能，先不说你能不能说服老板找100多人来做运维，最重要的是要是人参与的事情，就一定会犯错，比如rm敲错以后的懊悔，我估计所有做运维的读者都曾经经历过。

如果说几百台机器的代码发布工作，让人来一台一台干的话，我估计每天几十次的发布，会把这些人累趴下，很有可能键盘上的s和h是最早磨花的，因为要不停的“ssh”。

这么多服务器，天知道哪台服务器什么时候出问题、出什么问题，这时候，监控系统要出马了，出了问题第一时间通知到负责人，告知服务器的问题和出问题现场的服务器性能数据。

读到这里，有较真的读者要问了，“那CMDB是干嘛的？监控系统、发布系统、标准化系统我都明白是干什么的了，但CMDB好像跟前面几个都没什么关系。”其实，CMDB才是运维自动化的核心。为什么这么说，看我慢慢道来。

监控系统要发报警给负责人，那么某一台服务器谁是负责人，这个信息由谁来维护呢？有人会回答：“记在Excel里。”当我发布代码时，我想发布到www这个域名下，那么哪几台服务器是为www服务的呢？仍有人回答：“可以放Excel里。”标准化，要从一个机房搬一批机器到另一个机房，然后重装系统，给新业务服务。基础运维的同事怎么知道可以搬几台机器，去搬哪些机器呢？还是同样的声音：“记在Excel里。”可是，这些记在Excel里面的数据，由谁来维护更新呢？要知道，这些数据如果不更新，就是一堆垃圾。如果数据不准确，甚至还会有危险，比如把一台正在服务的服务器给关了。这里就更不要说Excel里面记错了、记录Excel的人离职了、两个Excel数据有冲突的问题了。

对了，CMDB就是这个Excel。当然，这个是对CMDB非常粗略的解释。CMDB除了是一个二维平面的表格外，还维护了“关系”这个概念，比如前面提到的服务器和域名的关系、服务器和负责人的关系、域名和域名之间的关系等。

1.3 监控系统在运维自动化中的角色

监控系统，是运维工程师和研发工程师的眼睛。它帮助工程师在第一时间发现网站的问题。

服务器的整个生命周期，都要和监控系统打交道。服务器上架，需要加入基础监控，比如 CPU 负载、内存等；当服务器上开始跑应用时，需要加入对应的应用监控，比如 Resin、MySQL 等；当服务器维护时，又要暂停这些报警，否则你明明在维护 MySQL，监控系统还给你报警说 MySQL 挂了。这些操作，手动去做非常麻烦，需要跟运维的其他系统共同协作，比如 CMDB 中一台机器上架了，那么监控系统自动加入监控。

监控系统是运维人员的眼睛，没有它，就没法知道系统运行状况，总不见得，我打开几十个终端，然后 while true 来显示 CPU 负载。撇开能不能看清楚的问题，得要多少显示器才能显示所有服务器的这些信息？

当发生问题时，监控系统要第一时间发出报警，报警中除了出问题的点，还可以有一些数据和简单的分析，比如当时一段时间的 CPU 负载等，以帮助接到报警的人员快速定位问题。

在出现故障以后进行问题分析时，还要靠监控系统。因为监控系统真实地记录了故障发生现场这台服务器的状况。运维工程师可以通过不同纬度的分析，找出问题的原因。

监控系统在运维自动化里的角色，可以用下面三点来概括。

- (1) 监控数据收集及可视化。
- (2) 异常数据报警。
- (3) 和其他系统协同工作。

1.4 监控系统的理想化模样

根据控监控系统在运维中的角色，理想的监控系统应该具有如下特点。

1. 监控数据收集及可视化。

- (1) 监控系统能够自定义监控的内容，可以自己写脚本来收集需要的数据。
- (2) 数据要保存在数据库中，这样以后需要的时候可以对这些数据进行分析计算。

（3）能够方便、快速地将监控加入到服务器上，不需要烦琐的操作。

（4）数据可视化不要很花哨，但要直观好用。

2. 异常数据报警。

（1）可以定义复杂的报警逻辑，可以做到 Item 之间的关联报警，而不是只能针对一个。

（2）报警需要被确认，让运维人员知道多少报警已经有人认领并开始处理了。

（3）报警方式要能够自定义，可以发邮件和短信，如果能够在 IM 上通知别人就更好了。

（4）报警内容要可以自行设置，在报警邮件中加入一些简单的分析，而不是让运维人员上服务器敲命令来获取基本的信息。

（5）报警后可以自动跑一些命令。这些命令可以是获取需要的信息，也可以是自动修复，比如重启服务等。

3. 和其他系统协同工作。

（1）有强大的 API 可以使用，可以让其他系统调用完成工作。

（2）监控数据是开放的，数据库中的数据结构不要太复杂，让人无从下手。

（3）监控可视化的图可以方便地引用，而不是要用一大串 JavaScript。

第 2 章

Zabbix简介

2.1 Zabbix发展现状

Zabbix 是一个非常强大的监控系统，其官方的说明是：Zabbix 是企业级的软件，被设计用来监控 IT 基础设施的可用性和性能。而我的看法是：它是一个能够快速搭建起来的、开源的监控系统。对于想快速可用的小型公司，Zabbix 自带的 Item 足够满足需求，通过简单的配置，可以在很短的时间内搭建起一套功能完善的报警系统。而对于中大型公司，Zabbix 也能很好地支撑，可以设定自定义的 Item，自动生成报表，有 API 和其他系统集成，数据库中有开放的数据可供分析。为什么暴雪的游戏（星际争霸、暗黑破坏神、魔兽争霸等）都这么经典？因为它们“易于上手，难于精通”。而 Zabbix 也一样，你可以非常容易地搭建起可用的 Zabbix，但要用好它，把它的潜能挖掘出来，需要花很大的力气。

使用 Zabbix 时，一般需要在被监控的服务器上安装 Zabbix Agent，Zabbix Server 会和 Zabbix Agent 进行通信，获取监控数据，这是 Zabbix 监控的一般模式。

Zabbix 从 2007 年开始流行，2013 年在国内开始流行起来。图 2-1 尾部的虚线部分是 Google Trends 对 Zabbix 的预测。图 2-2 是 Zabbix 在全球的区域热度。

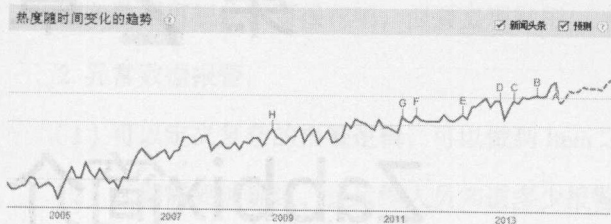


图2-1

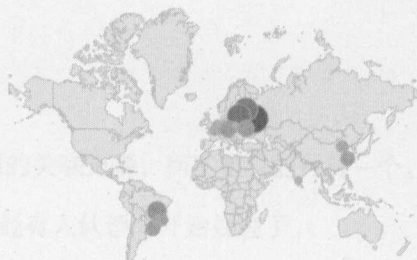


图2-2

Zabbix 目前的最新稳定版是 2.2.0，第一个版本是 2004 年 3 月发布的 Zabbix 1.0。Zabbix 从 2.0 开始，界面有了很大的变化，相比以前比较丑陋的 UI，变得有层次感了，并且增加了对于 JMX 的原生支持和 Linux 级别的自动发现。

Zabbix 目前的 Bug Fix 也比较正常，整个项目比较成熟，适合在公司中使用。笔者比较奇怪的是，Zabbix 的很多自定义监控脚本，都是通用的，但到目前为止，还没有一个平台，去共享这些脚本，在本书的最后一部分介绍开源和社区的地方，会提到笔者在 Github 上建立的，旨在分享 Zabbix 自定义监控脚本的项目。

2.2 选择Zabbix的理由

现在对应前文，看看 Zabbix 是否能满足理想中的监控系统的要求。

- ◎ 监控系统能够自定义监控的内容，可以自己写脚本来收集需要的数据——Zabbix 支持任何自定义的监控脚本，只要输出需要的值就可以。
- ◎ 数据要保存在数据库中，这样在需要的时候可以对这些数据进行分析计算——Zabbix 在数据库中的表结构虽然有些复杂，但逻辑很清晰。
- ◎ 能够方便、快速地将监控加入到服务器上，不需要烦琐的操作——Zabbix 有模板这一概念，可以方便地将一组 Item 进行统一操作。
- ◎ 数据可视化不要很花哨，但要用——Zabbix 每一个 Item 都可以看到其历史，Web 界面可拖动，界面友好。
- ◎ 可以定义复杂的报警逻辑，做到 Item 之间的关联报警，而不是只能针对一个——Zabbix 强大的 Trigger 定义，几乎可以满足所有规则组合。

- ◎ 报警需要被确认，让运维人员知道多少报警已经有人认领并开始处理了——Zabbix 对于报警，有 ACK 机制。
- ◎ 报警方式要能够自定义，可以发邮件、发短信，如果能够在 IM 上通知别人就更好了——Zabbix 支持邮件、Jabber。
- ◎ 报警内容要自己可设置，在报警邮件中加入一些简单的分析，而不是让运维人员上服务器敲命令来获取基本的信息——Zabbix 自定义了一套宏可以在报警邮件中引用，如果要更复杂的功能，可以通过远程调用命令实现。
- ◎ 报警后可以自动跑一些命令。这些命令可以是获取运维人员需要的信息，也可以是自动修复，比如重启服务等——在触发报警后，Zabbix 可以远程执行命令。
- ◎ 有强大的 API 可以使用，可以让其他系统来调用完成工作——Zabbix 支持 RestAPI，几乎所有的操作都可以通过 API 实现。
- ◎ 监控数据是开放的，数据库中的数据结构不要太复杂，让人无从下手——监控数据就在 Zabbix 数据库中，可以方便地进行分析。
- ◎ 监控可视化的图可以方便的引用，而不是要用一大串 JavaScript——Zabbix 使用 PHP 原生的绘图模块，如果要引用 Zabbix 的图表，只需要引用图表的 URL 即可，非常方便。

由此可见，Zabbix 满足我们所有对于监控系统的需求。是不是有些心动了呢？

2.3 Zabbix部分名词约定

Zabbix 有中文界面，但不是非常完善。我在写这本书的时候，一直在考虑，到底是用英文名词好，还是用中文名词好？如果使用英文名词，那么一部分英语不太好，或者因为各种原因使用中文界面的读者不容易理解；如果使用中文名词，有的地方官方翻译的又不是很通顺，不符合中国人的阅读习惯，比如“Discover”这个功能，Zabbix 翻译为“探索”和其本意相差太远。最后，我决定在本书中使用 Zabbix 原生的英语名词，然后在附录中会有一个中英文对照。因为，从现在的所有开源产品来说，英语仍是第一语言；在论坛中的交流，文档资料都是英语。要掌握一个开源产品，就必须了解它的基本英语的单词和意思，而且，Zabbix 本身并没有太多的英语需要理解。

虽然是基于 Zabbix 的中文翻译，但在开始学习 Zabbix 之前，先对一些 Zabbix 比较重要的东西和概念进行简单的解释。

- ◎ Zabbix Server : Zabbix 的控制中心, 收集数据、写入数据库都是它的工作。
- ◎ Zabbix Agent: 部署在被监控服务器上的一个进程, 负责和 Zabbix Server 交互, 执行命令。
- ◎ Host : 广义上的服务器, 大多数情况指代的是刀片机这类, 在少部分时间会指代包括交换机在内的, 被 Zabbix 监控的实体。
- ◎ Item : 对于某一个指标的监控, 对应的是 Items, 英文原意是“物品”。比如某台服务器的 CPU 负载就是一个 Item。
- ◎ Trigger : 一些逻辑规则的组合, 它有三个值: 正常、异常、未知。
- ◎ Action : 当 Trigger 符合某个值的时候, Zabbix 会进行的操作, 比如最常见的发邮件。

第 3 章

Zabbix安装

这一章主要介绍 Zabbix 的安装。总的来说，Zabbix 的安装不算困难：Zabbix Server 和 Zabbix Proxy 的安装步骤稍多，但 Zabbix Agent 安装非常简单。经过检验，在同样的系统中，编译安装完的 Zabbix Agent 可以复制到其他服务器中继续使用。

3.1 获取Zabbix

Zabbix Server 和 Zabbix Agent 都在一个压缩包中，可以到 <http://www.zabbix.com/download.php> 下载源码包，本书使用最新的稳定版的 Zabbix2.2.1 版本进行介绍。在下载界面中，除了 2.2 版本的源码外，还有下面几个可用的版本。

(1) Zabbix 2.2.X Packages：它们分别针对各个发行版做了打包，比如针对 RedHat 的 rpm，和针对 Debian 和 Ubuntu 的 deb 包。

(2) Pre-compiled Zabbix 2.2.0 (stable) agents are available for selected platforms：这个是已经编译好的，针对不同平台可以直接运行的包。

(3) Zabbix Appliance：这个是 Zabbix 基于 OpenSuSE 的镜像，它默认使用 MySQL 作为 Zabbix 的存储引擎。镜像中的系统，已经实现设置过，能够保证没有故障的运行。注意：这些镜像，适合在评估测试 Zabbix 的使用，目前还不适用于企业生产环境的使用。

Zabbix 是使用 C 语言编写的，笔者打算先花很小的篇幅介绍一下 Zabbix 源码包的目录结构。解压以后，一共有下面几个文件夹。

- ◎ bin : Windows 环境下使用。
- ◎ build : Windows 环境下使用。
- ◎ conf : Zabbix 配置文件的例子, 对于每一个参数都有说明。
- ◎ database : 数据库初始化文件。
- ◎ frontends : Zabbix 前端 PHP 代码。
- ◎ include : C 编译时需要的头文件。
- ◎ m4 : configure 脚本中 shell 代码的分装, 目的在于自动化生成 configure 和 Makefile。
- ◎ man : UNIX 使用手册 (UNIX Manual)。
- ◎ misc : 不同系统的 Zabbix 启动文件。
- ◎ src : Zabbix 源码, 不包括前端 PHP 代码。
- ◎ upgrades : 升级 Zabbix 时需要的, 供数据库 Scheme 升级使用。2.2 后已经废除, Zabbix 会自己更新, 但文件夹还是保留着。

3.2 Zabbix Server安装

安装前, 先看一下 Zabbix 对于硬件和软件的要求。Zabbix 官网对于小型规模、中型规模、大型规模、超大型规模这 4 种, 分别列举了例子, 如表 3-1 所示。

表3-1

规 模	CPU/ 内存	数据库	监控服务器数量
小型规模	PII 350MHz/256MB	SQLite	20 台
中型规模	AMD Athlon 3200+ 2GB	MySQL InnoDB	50 台
大型规模	Intel Dual Core 6400 4GB	RAID10 MySQL InnoDB 或者 PostgreSQL	1000 台以上
超大型规模	Intel Xeon 2xCPU 8GB	Fast RAID10 MySQL InnoDB 或者 PostgreSQL	10000 台以上

而对于我国互联网企业来说, 这点机器根本不算什么。如果说只是中小型规模, 一台非常一般的服务器也足够支撑所有的监控任务了。

3.2.1 Zabbix数据库配置

创建数据库时, Zabbix Server 需要初始化数据库的 Scheme 和其中的数据, Zabbix 代理 (后文会介绍) 只需要初始化数据库的 Scheme。对于 Zabbix Agent 来说, 不需要数据库的支

持。初始化数据库非常简单，下面会针对不同的数据库，分别详细说明如何配置（假设用户名 `username` 已经存在，权限也已经配置）。

对于 MySQL：

```
shell> mysql -u<username> -p<password>
mysql> create database zabbix character set utf8 collate utf8_bin;
mysql> quit;
shell> mysql -u<username> -p<password> zabbix < database/mysql/schema.sql
shell> mysql -u<username> -p<password> zabbix < database/mysql/images.sql
shell> mysql -u<username> -p<password> zabbix < database/mysql/data.sql
```

对于 PostgreSQL：

```
shell> psql -U <username>
psql> create database zabbix;
psql> \q
shell> cd database/postgresql
shell> psql -U <username> zabbix < schema.sql
shell> psql -U <username> zabbix < images.sql
shell> psql -U <username> zabbix < data.sql
```

对于 Oracle：

```
shell> sqlplus zabbix/password@host/ORCL
sqlplus> @database/oracle/schema.sql
sqlplus> @database/oracle/images.sql
sqlplus> @database/oracle/data.sql
```

这些 SQL 文件都在 `database` 文件夹中，需要注意的是，我们一定要把数据库设置为 UTF8 编码。另外，如果是使用 Zabbix Proxy，只需要导入 `schema.sql` 就行了，`images.sql` 和 `data.sql` 都不必导入。

3.2.2 安装 Zabbix Server

初始化 Zabbix 数据库后，本节讲解 Zabbix Server 的安装。先从前文说到的地方把 Zabbix 2.2 版本的源码包下载下来，然后解压，进行以下操作。

1. 创建用户

对于所有的 Zabbix 进程来说，一个非 root 用户是必须的。当 Zabbix 进程以那个身份启动的时候，它就会一直以那个进程的身份运行。

当使用 root 用户启动 Zabbix 进程的时候，它会切换到 zabbix 用户，所以 zabbix 用户一定要存在。这里我们首先建立一个 zabbix 用户组，然后添加 zabbix 用户到里面：

```
>groupadd zabbix
>useradd -g zabbix Zabbix
```

大家注意，Zabbix 的 PHP 前端的运行是不需要 zabbix 这个用户的。Zabbix 官网建议，如果 Zabbix Server 和 Zabbix Agent 是在一台机器上运行的话，最好使用两个用户，来隔离权限，这样有什么好处呢？如果说都使用同一个用户跑，那么 Zabbix Agent 可以去修改 Zabbix Server 的配置文件，而且对于任何一个 Zabbix 的管理员，都可以轻松获取 Zabbix Server 的敏感信息，比如 Zabbix 数据库的 IP，用户名，密码。我见过很多 Zabbix 的 PHP 前端放在公网上，这样我相当于获取了 Zabbix Server 的读写权限（使用 system.run，可以简单理解为远程执行命令）。

另一点需要大家注意的就是，不要把 Zabbix 运行在 root 身份或者任何拥有管理员权限的身份，这样有很大的风险。这样进入了 Zabbix，相当于获取了所有服务器的 root 权限。非常恐怖，你能想象我在所有服务器上执行 `rm -rf/` 的结果吗？

2. 编译源代码

在使用 configure 编译 Zabbix 源代码的时候，需要指定要安装的是 Zabbix Server，还是 Zabbix 代理，或是 Zabbix Agent，还需要指定使用的数据库类型，此外，Zabbix 还支持很多在编译时的配置。具体可以使用 `./configure --help` 来查看。比如，我们要安装 Zabbix Server 和 Zabbix Agent，使用 MySQL 作为数据库，就可以这么安装：

```
./configure --enable-server --with-mysql
```

这里有几个需要注意的地方：

- ◎ 如果需要监控虚拟机的功能，需要加 `--with-libxml2`
- ◎ 使用 `--prefix=/home/of/zabbix` 可以指定 Zabbix 的安装目录

比如我们要在这台机器安装 server, agent, 并且使用 mysql 数据库, configure 的时候就是这样：

```
shell> ./configure --enable-server --enable-agent --with-mysql
```

一般来说,我们会开启 SNMP 监控,那么还要加上 `--with-snmp`。

如果想一步到位,可以像如下这样:

```
./configure --enable-server --enable-agent --with-mysql --with-net-snmp
--with-libcurl --with-libxml2 --with-openipmi --with-unixodbc --prefix=/
apps/svr/zabbix.
```

编译源码是最容易发生错误的,常见的就是 “Not found XXX library”,下面我们来针对各种常见情况一个一个解决。

- ◎ Not found mysqlclient library : 安装 `mysql-devel`
- ◎ LIBXML2 library not found : 安装 `libxml2-devel`
- ◎ Curl library not found : 安装 `curl-devel`
- ◎ cannot use unixODBC library : 安装 `unixODBC-devel`
- ◎ Invalid Net-SNMP directory - unable to find net-snmp-config : 安装 `net-snmp-devel`
- ◎ Invalid OPENIPMI directory - unable to find ipmiif.h : 安装 `OpenIPMI-devel`

当我们看到如图 3-1 所示的界面,就是编译成功了。



```
*****
*               Now run 'make install'               *
*                                                     *
*               Thank you for using Zabbix!           *
*               <http://www.zabbix.com>              *
*****
```

图3-1

3. 安装

这一步非常简单,就是 `make install`。默认情况下(即前一步没有指定 `prefix`), `zabbix_server`、`zabbix_proxy` 和 `zabbix_agentd` 会安装在 `/usr/local/sbin`,而客户端的 `zabbix_get` 和 `zabbix_sender` 会在 `/usr/local/bin`。

4. 修改配置文件

需要修改的就是 Zabbix Server 连接数据库的 IP、用户名和密码。

5. 运行 Zabbix Server

```
shell> /usr/local/sbin/zabbix_server
```

3.2.3 安装Zabbix Web前端

Zabbix 前端配置并不困难，在笔者刚接触 Zabbix 的时候，对于 PHP 一窍不通，安装 ZabbixWeb 前端的问题主要在 PHP 上。其实，安装了一次以后，就有经验了，觉得也不是困难的事情。

首先安装 PHP 和 Apache：

```
sudo yum install php53
```

```
sudo yum install httpd
```

然后可以正式安装 Zabbix Web 前端了，具体如下。

（1）Zabbix 前端和 Zabbix 可以是分开的两台机器，只要求 Zabbix 前端和 Zabbix Server、Zabbix 数据库网络连通即可。

（2）将 Zabbix 源代码文件夹中的 frontend 中的内容复制到容器对应目录，在浏览器中打开 Zabbix 前端，如果一切正常的话，已经可以看到界面了。如果没有看到界面，而是看到一堆文件，那说明 PHP 没有安装正确，请按照前面的步骤正确安装 PHP。

（3）然后就是前端配置中最容易出错的一步了。从前端我们可以看到 Zabbix 前端对于系统的要求。根据 Zabbix 官网，Zabbix 对于 PHP 的要求如表 3-2 所示。

表3-2

要 求	最小值	说 明
版本	5.3.0	
memory_limit	128MB	
post_max_size	16MB	
upload_max_filesize	2MB	
max_execution_time	300 秒	
max_input_time	300 秒	
session.auto_start	禁止	
数据库支持	根据需要	比如你使用 MySQL，就一定要安装 PHP 对于 MySQL 的支持，其他的 Oracle 等类似
bcmath		php-math
mbstring		php-mbstring
sockets		php-net-socket。这个是为了支持用户脚本（在 Action 中）

续 表

要 求	最小值	说 明
gd	2.0 或者更高	php-gd, 它是 PHP 对于各种图片的支持。PHP GD 一定要支持 PNG (--with-png-dir), JPEG (--with-jpeg-dir 和 FreeType2 (--with-freetype-dir)
libxml	2.6.15	php-xml 或者 php5-dom
xmlwriter		php-xmlwriter
xmlreader		phpxmlreader
ctype		php-ctype
session		php-session
gettext		php-gettext

根据图 3-2 所示, 界面中的 fail 提示在实际界面中是红色的, 我们需要修改 PHP 配置, 一般是在 /etc/php.ini 文件中。我们根据图中关于 PHP option 的提示, 把每一个不符合要求的都修掉, 然后重启 httpd。

关于不是 PHP option 的 fail 提示, 则是 Zabbix 需要的数据库依赖, 这里以最常用的 MySQL 为例:

```
shell> sudo yum install php53-mysql php53-bcmath php53-mbstring php53-gd php53-xml
```

解决完全部问题后, 就可以单击图 3-2 中原本是灰色的 Next 了。

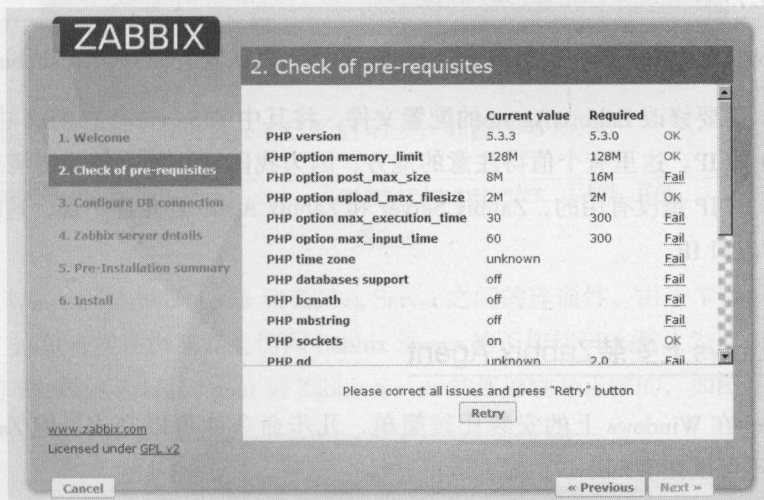


图3-2

（4）输入数据库相关配置，单击 Test connection，如果提示 Can't connect to local MySQL server through socket '/tmp/mysql.sock'，则需要将服务器上的 mysql.sock 在 /tmp 下做一个软连接：

```
shell> ps -ef | grep mysql.sock  
shell> ln -s /var/lib/mysql/mysql.sock /tmp/mysql.sock
```

（5）输入 Zabbix Server 配置。

（6）下载 Zabbix 前端的配置。这个配置文件是根据前面输入的数据库参数、Zabbix Server 参数生成的。下载后放在提示的位置即可。

（7）完成安装。默认的用户名密码是 Admin 和 zabbix，切记登录的第一件事就是把这个默认密码给改了。

3.3 Zabbix Agent安装

3.3.1 UNIX/Linux上安装Zabbix Agent

在 UNIX/Linux 的机器上，安装 Zabbix Agent 非常简单，就像其他 Linux 源码一样，configure，make install 即可。命令如下：

```
cd $ ZABBIX_SRC_DIR configure--enable-agent make install
```

和安装 Zabbix Server 一样，如果不指定 prefix 的话，默认安装在 /usr/local/sbin 下。

安装完后，需要修改 Zabbix Agent 的配置文件，将其中的 Server=127.0.0.1 中的 IP 地址改为 Zabbix Server 的 IP。这里有个值得注意的地方，因为我国错综复杂的网络情况，有时这里写 Zabbix Server 的 IP 是没有用的，Zabbix Server 和 Zabbix Agent 还是连不通，这时就需要写成 Zabbix Server 的出口 IP。

3.3.2 Windows上安装Zabbix Agent

Zabbix Agent 在 Windows 上的安装比较简单，几步命令就可以完成，但 Zabbix Server 和 Zabbix Proxy 都不可以在 Windows 机器上运行。

Zabbix Agent 的 Windows 版本就在其安装目录下的 bin 文件夹中，其中分了 win32 和 win64

版本,大家根据情况自行选择。在C盘根目录下有 zabbix_agentd.conf,注意这个 conf 和 Linux 下的 conf 是不一样的,不能混用。从 Zabbix 安装目录下的 conf 文件夹中找到 zabbix_agentd.win.conf,重命名为“zabbix_agentd.conf”后将其放到C盘根目录下,并配置相关的参数。下面来看看在 Windows 系统安装 Zabbix Agent 的详细步骤。

(1) 选择“cd c:\PATH\TO\ZABBIX\bin\win64”路径。

(2) 选择“zabbix_agentd.exe -install”,安装服务,成功则提示如图 3-3 所示。

```
C:\zabbix-2.2.0\bin\win64>zabbix_agentd.exe --install
zabbix_agentd.exe [2680]: service [Zabbix Agent] installed successfully
zabbix_agentd.exe [2680]: event source [Zabbix Agent] installed successfully
```

图3-3

这个时候 Zabbix Agent 已经在 Windows 中生成一个服务了,可以从“开始”→“管理工具”→“服务”中看到,如图 3-4 所示。

Zabbix Agent Provides system monitoring 自动 本地系统

图3-4

install 这一步,默认是调用 c:\zabbix_agentd.conf 的,如果你的 conf 不在这里,可以使用 config 来指定。

(3) 选择“zabbix_agentd.exe -start”,启动 Zabbix,启动正常则显示如图 3-5 所示。

```
C:\zabbix-2.2.0\bin\win64>zabbix_agentd.exe --start
zabbix_agentd.exe [10248]: service [Zabbix Agent] started successfully
```

图3-5

从服务中也可以看到 Zabbix Agent 已经启动了,如图 3-6 所示。

Zabbix Agent Provides system monitoring 已启动 自动 本地系统

图3-6

现在测试 Windows Zabbix Agent 和 Zabbix Server 之间的连通性。由于 Windows 是没有自带 telnet 工具的,这里直接在浏览器上访问 Zabbix Server 的工作端口(默认为 10051),如果显示“OK”,则说明 Windows Zabbix Agent 到 Zabbix Server 的连通性是正常的,如图 3-7 所示。

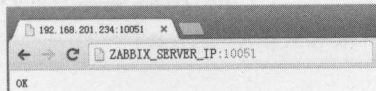


图3-7

3.4 测试Zabbix Agent和Zabbix Server运行

在把 Zabbix Server、Zabbix 前端和 Zabbix Agent 都安装好后，这一节来检查 Zabbix Server 和 Zabbix Agent 是不是能连通。

方法很简单，Zabbix Agent 是在 10050 端口接收 Zabbix Server 命令的，我们只要测试 Zabbix Server 端能否和 Zabbix Agent 连通即可：

```
telnet ZABBIX_AGENT_IP 10050
```

这里的 ZABBIX_AGENT_IP 指的是 Zabbix Agent 的 IP。

3.5 配置文件详解

Zabbix 的配置虽不十分复杂但也不少，有的配置参数，笔者也是使用了好久才知道是干什么的。在这一章中，已经基本介绍了 Zabbix 的各个组件，说明了它们的作用。接下来，笔者会针对 Zabbix Server、Zabbix Agent 和 Zabbix Proxy 的配置文件，对其中的每个参数进行介绍，不求深入，希望能让大家知道每个参数是干什么用的。

3.5.1 zabbix_server.conf

从名字就看的出，这个是 server 的配置文件，它的参数作用（参数的范围不在此说明，配置文件中），如表 3-3 所示。

表3-3

参数名称	是否必须	默认值	解 释
NodeID	否	0	在 Master-Child 的分布式架构中，这个 ID 是唯一标识 Zabbix Node 的号码
ListenPort	否	10051	Trapper 类型 Item 监听的端口
SourceIP	否	空	在连接其他服务器时，使用的本机 IP 地址
LogFile	否	如果不设置，会使用 syslog	存放 Zabbix Server 日志文件的地方，需要指定

续 表

参数名称	是否必须	默认值	解 释
LogFileSize	否	1	单个最大日志文件大小，超过则会日志轮转。设置为 0 则永远不会日志轮转。单位是 MB
DebugLevel	否	3	定义打印的日志等级。“0”为打印日志，“1”打印重要的错误日志，“2”打印错误信息，“3”打印警告信息，“4”打印调试信息。“4”会产生大量的日志，一般在排查问题时使用
PidFile	否	/tmp/zabbix_server.pid	记录 Zabbix Server pid 的文件位置
DBHost	否	localhost	Zabbix Server 数据库的位置，如果设置为 localhost，那么端口会使用 MySQL 的端口，如果设置为空，那么会使用 PostgreSQL 的端口
DBName	是	空	数据库名字。对于 SQLite3，必须定义数据库文件的位置，而数据库用户名和密码不需要
DBScheme	否	空	仅对 IBM DB2 生效
DBUser	否	空	数据库用户名，对 SQLite 无效
DBPassword	否	空	数据库密码，对 SQLite 无效
DBSocket	否	/tmp/mysql.sock	MySQL socket 文件的路径
DBPort	否	3306	MySQL 端口
StartPollers	否	5	pollers 进程数，poller 可以简单理解为 Zabbix 工作的一个 worker
StartIPMIPollers	否	0	IPMI pollers 进程数
StartPollersUnreachable	否	1	检查 unreachable hosts（包括 IPMI）的进程数
StartTrappers	否	5	Trappers 的进程数。Trappers 接收其他 host 用 zabbix_sender、active agents、active proxies 和 child nodes 发送的数据。至少需要一个 Trapper 进程用来在前端显示 Zabbix Server 可用性
StartPingers	否	1	用于 discover 的 discoverer 的进程数
StartHTTPPollers	否	1	用于 HTTP 检查的进程数
StartTimers	否	1	Timers 的进程数。Timers 进程用于处理基于时间的 Triggers 中的 function 和 maintenance 功能。只有第一个 Timer 进程处理 maintenance 时间
JavaGateway	否	空	Zabbix Java gateway 使用的 IP 或者 hostname。当 Java pollers 启动时有效
JavaGatewayPort	否	10052	Java gateway 使用的端口

续 表

参数名称	是否必须	默认值	解 释
StartJavaPollers	否	0	Java pollers 的进程数
StartVMwareCollector	否	0	VMware pollers 的进程数
VMwareFrequency	否	60	Zabbix 从 VMware 获取监控值的频率，单位是秒
VMwareCacheSize	否	8MB	VMware 的缓存，存储 VMware 数据的共享内存大小。只有当 VMware collectors 启动时生效
SNMPTrapperFile	否	/tmp/zabbix_traps.tmp	SNMP 设备在将数据发送到 server 前会将 SNMP 数据存在文件中。必须和在 zabbix_trap_receiver.pl 或者 SNMPTT 配置文件中的配置相同
ListenIP	否	0.0.0.0	设定的是用逗号分隔的 IP 列表，Trappers 监听的 IP
HousekeepingFrequency	否	1	Zabbix 执行 Housekeeper 的频率
MaxHousekeeperDelete	否	500	在 Zabbix 数据库中，有一张“housekeeper”表，里面记录了 Housekeeper 要执行的“任务”，格式为“housekeeperid”，“tablename”，“field”，“value”。在一次执行 Housekeep 的过程中，最多删除这里定义的数量。SQLite3 会忽略这个参数，它会一次删除所有相关的行，如果这里设置为 0，那么就相当于没有限制，请谨慎
SenderFrequency	否	30	Zabbix 发送报警的时间间隔
CacheSize	否	8MB	存储 Host、Item 和 Trigger 数据的内存空间
CacheUpdateFrequency	否	60	将配置信息同步到内存中的频率
StartDBSyncers	否	4	将数据同步到数据库的 DB Syncers 进程数
HistoryCacheSize	否	8MB	存储 History 数据的内存大小
TrendCacheSize	否	4MB	存储 Trends 数据的内存大小
HistoryTextCacheSize	否	16MB	存储 character，text 和 log 类型的 History 数据的内存大小
ValueCacheSize	否	8MB	History 数据缓存在内存中的内存大小。如果设置为 0，则不缓存
NodeNoEvents	否	0	这个参数对 Master-Child 架构有效，设置为“1”，那么本地 event 不会发送到 master 节点。这个参数只对本节点有效，不会影响其他节点，即如果这个节点有子节点，不会影响其子节点
NodeNoHistory	否	0	同“NodeNoEvents”，这里定义的是本地的 History 数据
Timeout	否	3	Zabbix 等待 Agent、SNMP 设备或者自定义脚本的执行时间
TrapperTimeout	否	300	Trapper 处理新数据的超时时间

续 表

参数名称	是否必须	默认值	解 释
UnreachablePeriod	否	45	当一个 Host 保持 unreachable 状态后多久将其标记为 unreachable 状态
UnavailableDelay	否	60	当 Host 为 unavailable 状态时, 检查 Host 的 availability 的频率
UnreachableDelay	否	15	当 Host 为 unreachable 状态时, 检查 Host 的 availability 的频率
AlertScriptPath	否	\${datadir}/zabbix/alertscripts	自定义报警脚本的位置
ExternalScripts	否	\${datadir}/zabbix/externalscripts	自定义监控脚本的位置
FpingLocation	否	/usr/sbin/fping	Fping 的位置。Fping 可执行文件的 owner 要设置为 root, 并且设置 suid
Fping6Location	否	/usr/sbin/fping6	同上。如果 Fping 可以处理 IPv6, 那么可以留空
SSHKeyLocation	否	空	使用 SSH 检查和 action 所需要的 SSH 公钥、私钥位置
LogSlowQueries	否	0	记录查询 Zabbix 数据库的慢查询, 单位是毫秒。只有当 DebugLevel 设置为 3 或者 4 才会生效。如果设置为 0, 则不记录慢查询。这个对 Zabbix 性能差是非常好的 debug 方式
TmpDir	否	/tmp	临时文件目录
StartProxyPollers	否	1	被动 Proxy 的 poller 进程数
ProxyConfigFrequency	否	3600	Zabbix Server 将配置信息同步到 Proxy 的频率。这个参数只对被动的 Proxy 生效
ProxyDataFrequency	否	1	Zabbix Server 请求 Proxy 历史数据的频率。这个参数只对被动的 Proxy 生效
AllowRoot	否	0	是否允许 Server 以 “root” 身份运行。“0” 表示不允许, “1” 表示允许。如果不允许以 root 运行, 并且 Zabbix 以 root 身份运行, 那么 server 会尝试切换到 zabbix 用户。如果 server 使用一般用户启动, 这个参数没有效果
Include	否	空	指定存放了设置自定义监控的文件位置
LoadModulePath	否	\${libdir}/modules	loadable 组件的位置
LoadModule	否	空	需要 Server 载入的 loadable 组件, 格式为 LoadModule=<module.so>

3.5.2 zabbix_agentd.conf

配置参数如表 3-4 所示。

表3-4

参数名称	是否必须	默认值	解 释
PidFile	否	/tmp/zabbix_agentd.pid	记录 Zabbix Agentd Pid 的文件
LogFile	否	如果不设置, 会使用 syslog	存放 Zabbix Server 日志文件的地方, 需要指定
LogFileSize	否	1	单个最大日志文件大小, 超过则会日志轮转。设置为 0 则永远不会日志轮转。单位是 MB
DebugLevel	否	3	定义打印的日志等级。“0”为打印日志,“1”打印重要的错误日志,“2”打印错误信息,“3”打印警告信息,“4”打印调试信息。“4”会产生大量的日志,一般是在排查问题时使用
SourceIP	否	空	对外发起网络连接时使用的 IP
EnableRemoteCommand	否	0	Remote Command 指的是来自 Zabbix Server 的命令这个选项控制 agent 是否允许执行这些命令。“0”表示不允许,“1”表示允许
LogRemoteCommands	否	0	是否将 remote command 记录下来,作为 warning 级别的日志
Server	否	127.0.0.1	Zabbix Server 的 IP 或者 hostname。当有多个时,可以使用逗号分隔。Zabbix Agent 只会接受来自这些 IP 或者 hostname 的连接。如果系统支持 IPv6, 就可以使用 “::127.0.0.1” 和 “::ffff:127.0.0.1”
ListenPort	否	10050	Zabbix Agent 监听的端口
ListenIP	否	0.0.0.0	Zabbix Agent 监听的 IP
StartAgents	否	3	处理被动检查的 zabbix_agentd 进程数。如果设置为 0, 那么这个 Agent 的被动检查功能将禁止, Agent 不会监听在所有的 TCP 端口
ServerActive	否	空	用于主动检查的 IP (或者 hostname) 和端口。如果不指定端口, 会使用默认端口。如果既使用 IPv6, 又要指定端口, 那么需要将 IPv6 的 IP 地址写在方括号中, 比如 [::1]:30051。如果这个参数为空, 那么主动规则将被禁止。如果有多个, 请用逗号分隔
Hostname	否	空	运行 Zabbix Agentd 的 hostname。必须与 server 中配置的 hostname 相同
HostnameItem	否	system.hostname	如果 Hostname 为空, 会使用这个参数定义的 key 的值作为 Hostname, 比如默认的 system.hostname 这个 Zabbix 自带的 Key
HostMetadata	否	空	这个参数是在 host auto-registration 中使用的。如果超过 255 个字符, Agent 会报错。如果没有设置, 那么会从 HostMetadataItem 这个参数中获取

续 表

参数名称	是否必须	默认值	解 释
HostMetadataItem	否	空	类似 HostnameItem, 使用一个 key 的值作为 host metadata 的值
RefreshActiveChecks	否	120	主动检查项目列表刷新的时间
BufferSend	否	5	在 buffer 中缓存多少秒的数据
BufferSize	否	100	在内存 buffer 中缓存数据的最大个数, 当 buffer 中的数据个数达到最大值后, 会将数据发送到 server 或者 proxy
MaxLinesPerSecond	否	100	对于 “log” 和 “logrt” 类型的 item, 这个参数定义了每秒发送给 server 或者 proxy 的最大行数。如果在 item 的 “maxlines” 中设置过, 那么会覆盖这个参数
Alias	否	空	对于 parameter 设置别名。这个对非常长或者非常复杂的 parameter 名字特别有效
Timeout	否	空	处理数据的超时时间
AllowRoot	否	0	是否允许 Agent 以 “root” 身份运行。“0” 表示不允许, “1” 表示允许。如果不允许以 root 运行, 并且 Zabbix 以 root 身份运行, 那么 Agent 会尝试切换到 zabbix 用户。如果 Agent 使用一般用户启动, 这个参数没有效果
Include	否	空	指定存放了设置自定义监控的文件位置
UnsafeUserParameters	否	0	在用户自定义的 parameters 中, 所有的参数都会传递给脚本
UserParameter	否	空	用户自定义监控脚本的 parameter。parameter 意为 key 和脚本的对应关系。格式为 UserParameter=<key>,<shell command>
LoadModulePath	否	\$(libdir)/modules	loadable 组件的位置
LoadModule	否	空	需要 server 载入的 loadable 组件, 格式为 LoadModule=<module.so>

第 4 章

监控第一台Host

在第 3 章，我们搭建了 Zabbix Server 和 Zabbix 前端，同时也在服务器上安装了 Zabbix Agent。在这一章，我们把一台服务器加入 Zabbix 监控，从真实的需求出发，实现 Zabbix 对于服务器的监控、报警。通过本章的实践，希望读者能够掌握 Zabbix 的基本使用方法。

4.1 Host在监控系统中的活动

在监控系统中，核心对象是服务器。一般来说，监控都是针对服务器而言的，而对于服务器来说，核心是其 Item。有了 Item，就可以发现监控指标的异常，然后就是要报警，这应该是每一个运维工程师每天都要面对的情况。无论公司大小、运维有多少服务器，报警邮件一定是运维工程师最不缺的东西。在解决问题的时候，要查看服务器的性能数据，从而分析定位问题。比如，当发现服务器负载突然变高时，可能需要看磁盘 I/O 是不是升高，这时需要查看各个历史数据，看其是否和出问题时的时间点吻合。要是发现磁盘 I/O 在 CPU 负载突然升高时也突然升高，那很有可能就是磁盘 I/O 造成的 CPU 负载高。

因此，服务器在监控系统中的活动，主要有以下三点：

- ◎ 监控数据指标
- ◎ 数据异常时报警
- ◎ 监控数据的数据可视化

这是监控系统最重要的组成，其他都是基于这三点的。

4.2 添加一个用户

首先,要定义一个使用 Zabbix 的用户,默认这时已存在一个 zabbix 用户。

如图 4-1 所示,在菜单上单击“Administration”→“Users”,再单击“create user”,根据屏幕上的提示输入对应的信息。在登录 Zabbix 时,如果连续 5 次输入密码错误,系统会锁定 Zabbix,以防止暴力破解。所以新建用户时,一定要记住密码。

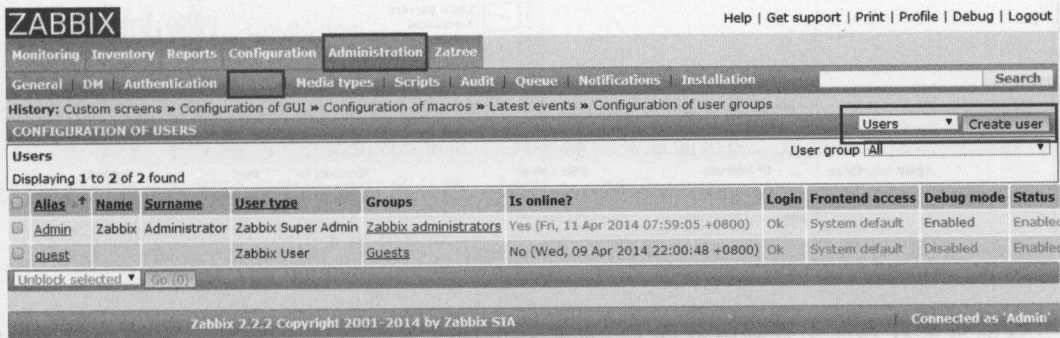


图4-1

在新建一个用户时,默认是没有任何媒介的,这里单击“Media”标签,新建一个媒介,“Type”选择邮件,然后输入这个用户接收 Zabbix 报警时使用的邮箱即可,本书中我们使用 frank@zabbix.com 为例。

其他的一些设置不需要修改。添加用户后,就可以添加监控了。

4.3 把服务器加入Zabbix监控

一台服务器,在 Zabbix 中被称为 Host,在本书中,就用服务器来称呼。笔者曾考虑将本书和后面的“添加一个 Item”小节合并到一起,因为把服务器加入 Zabbix 监控是个非常简单、非常“微不足道”的步骤,但考虑再三后,还是把它独立出来。因为“服务器”是 Zabbix 非常重要的概念,它关系到整个 Zabbix 的资源模型。关于 Zabbix 资源模型,会在这一章的结尾部分介绍,有兴趣的读者可以先跳过去看,然后再回过头来阅读这一小节。

如图 4-2 所示,在 Zabbix 前端选择“Configuration”→“Hosts”→“Create host”,在跳转的界面中,输入服务器的名字和 IP,其他的配置目前还不需要修改,保持默认值。最后选择保存。

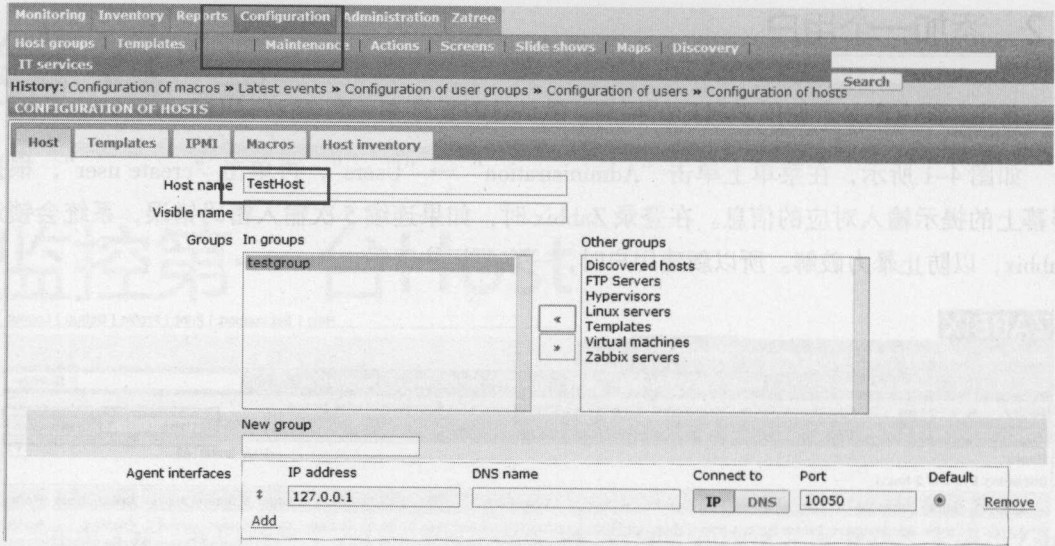


图4-2

这时就能在界面中看到服务器了。如果在可用性一栏是一个红色 Z，那么说明 Zabbix Server 和这台机器的 Zabbix Agent 之间的通信有问题，可以把鼠标放上去，查看具体的问题。如果 Z 图标是灰色的，说明很久没有更新数据了，应该检查 Zabbix Server 是不是正常，或者尝试刷新页面，如图 4-3 所示。



图4-3

4.4 添加Item

Item 是服务器上的监控项，没有 Item 的服务器就像个空壳子，没有任何用处。这一节，讲解如何给服务器添加一个 CPU 负载的 Item。

(1) 从右上角的搜索框找到服务器，如图 4-4 所示。

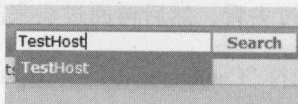


图4-4

(2) 单击 Items, 会出现 Item 的管理界面。在图 4-5 中, Items 后面的数字“0”, 表示目前这台机器上还没有添加 Item, “Triggers (0)” 的意思与此相同。



图4-5

(3) 单击右侧的“Create item”。

(4) “Name” 栏中输入“CPU Load”, 如图 4-6 所示。

(5) “Key” 栏中输入“system.cpu.load”, 其具体含义后面再说, 这里可以简单地理解为指定监控的内容。这里代表监控 CPU 的负载, 如图 4-6 所示。

(6) 在“Type of information” 中选择“Numeric (float)”, 表示 Item 的数据类型, 如图 4-6 所示。

Name

Type

Key

Host interface

Type of information

图4-6

现在服务器已经添加完成。可能有读者会问, 添加一个 Item 都这么麻烦, 那以后服务器多了, 怎么管理呢? Zabbix 有监控模板的概念, 就是一组 Item 在一起, 可以将一个监控模板和一台服务器关联, 那么监控模板上的所有 Item, 包括其他东西 (Trigger、Graph 等), 都会添加到这台服务器上。而关于大量机器的自动化监控, 在 Template 章节会有讲解, 这里不展开了。

4.5 添加Trigger

Trigger, 即触发器, 当出现某些情况时, 它会发出某些提示行为。在 Zabbix 里, 可以对一个 Trigger 定义一些触发的条件, 比如最常见的某个 Item 的值超过了某个阈值, 然后定义这个 Trigger 触发后应该干什么。

Trigger 是 Zabbix 报警的核心之一, 它的功能强大而复杂。这一节为之前添加的 Item 关联

一个 Trigger，当 CPU 负载超过某个阈值时，会触发这个 Trigger。

除了从右上角的搜索框找到服务器，也可以从任意位置使用菜单栏找到它，路径是“Configuration”→“Hosts”。然后单击“Triggers”进入配置 Trigger 的界面，单击右上角的“Create trigger”，添加一个新的“Trigger”。

在跳转的页面中，在“Name”中输入 Trigger 的名称，这里把它叫作“CPU Load is too high”；在“Expression”中，输入“{TestHost:system.cpu.load.last()}>2”。这里是使用 Zabbix 的语法定义了一个表达式，意思是刚刚添加的服务器的 CPU 负载的 Item，获取的数据大于 2。目的是让它跑起来。目前只需要定义这两个就可以了。为了测试，这里把“2”改成“0”，让它马上能报警。再把下方的“Severity”选择为“High”，代表一个较为严重的报警，如图 4-7 所示。

The screenshot shows the Zabbix Trigger configuration form. The fields are as follows:

- Name:** CPU Load is too high
- Expression:** {TestHost:system.cpu.load.last()}>0
- Expression constructor:** (Link)
- Multiple PROBLEM events generation:** ☐
- Description:** (Empty text area)
- URL:** (Empty text field)
- Severity:** Not classified, Information, Warning, Average, **High**, Disaster
- Enabled:** ☒
- Buttons:** Save, Cancel

图 4-7

单击菜单栏中的“Monitoring”，再单击“Trigger”，就可以看到刚刚定义的 Trigger。这一行最前面有一个绿色的标签“OK”，表示刚刚定义的表达式没有被触发，即 Host 这台服务器的 CPU 负载没有超过设定的阈值 2。如果是一个红色的“PROBLEM”，就说明 CPU 负载超过阈值了。

4.6 设置Action

Action, 即报警动作是对于 Zabbix Event 的响应。它支持下面这些 Zabbix Event。

- Trigger Event : Trigger 从“正常”变为“异常”等。
- Discovery Event : 在“Configuration”→“Discovery”中配置的 Discovery rule 生效时。
- (Auto registration Event) 服务器自动发现事件 : Zabbix 自动发现了新的服务器。
- Internal Event : Item 出现问题不可用或者 Trigger 处于未知状态“Internal”的意思是内部的, 表示 Zabbix 本身出的问题。

这里用最普通的, 即 Trigger 事件的异常来配置一个报警动作。

在运维工作中, 对于服务器的问题, 我们希望马上就能知道, 但又不可能时时刻刻盯着每一个监控数据来检查是否有问题, 所以就需要一个机制, 使 Item 出现问题时, Zabbix 可以通知我们。使用前一节的 Trigger, Zabbix 可以捕捉到 Item 的异常, 而在这一节, 我们将学习当出现问题的时候, 如何让 Zabbix 发送邮件给我们。

要 Zabbix 发送邮件, 首先需要配置 Zabbix 使用的邮件服务器。从菜单选择“Administration”→“Media”, 可以看到 Zabbix 已经默认定义了三种媒介: Email 是邮件; Jabber 是 XMPP; Extensible Messaging and Presence Protocol 是以 XML 为基础的开放式即时通信协议, 这个国内用的很少; SMS 是短信息, Zabbix 默认的 SMS 服务需要在机器上安装一个发送短信的硬件设备, 一般不会这么做。如果要发短信, 在书的后半部分, 会介绍微信报警的解决方案。

单击 Email, “Name”是我们给这个媒介取的名字; “Type”选择“Email”; SMTP 是邮件服务器, SMTP helo 是一个域名, 一般是邮件服务器 @ 符号后面的, 比如 @google.com; SMTP email 是发送报警邮件的邮箱。

报警邮件已经设置好了, 接下来要新建一个 Action。在 Zabbix 中, 新建一个发送邮件类型的报警动作, 非常简单。从菜单进入, 选择“Configuration”→“Actions”, 单击“Create action”, Action 名字输入“CPU Load is too high”。在 Condition 标签页中, 需要配置 Action 触发的场景, 这里选择“Trigger name like CPU Load”, 那么整个 Condition 有三个条件, 如图 4-8 所示。

Conditions	Label	Name	Action
	(A)	Maintenance status not in <i>maintenance</i>	Remove
	(B)	Trigger value = <i>PROBLEM</i>	Remove
	(C)	Trigger name like <i>CPU Load</i>	Remove

图4-8

图中的(A)表示服务器不在维护状态中, Trigger的状态是PROBLEM, 并且Trigger的名字是“CPU Load”。通过这些条件, 这个Action就和我们刚刚建立的Trigger关联起来了。

接下来就是最后一步了, 选择“Operation”标签, 在“Operation type”中选择“Send message”, 在“Send to Users”中选择“frank”。单击“Add”按钮添加Operation。再单击“Save”保存, 如图4-9所示。

Step	From	<input type="text" value="1"/>						
	To	<input type="text" value="1"/> (0 - infinitely)						
	Step duration	<input type="text" value="0"/> (minimum 60 seconds, 0 - use action default)						
Operation type	<input type="text" value="Send message"/>							
Send to User groups	<table><tr><th>User group</th><th>Action</th></tr><tr><td>Add</td><td></td></tr></table>		User group	Action	Add			
User group	Action							
Add								
Send to Users	<table><tr><th>User</th><th>Action</th></tr><tr><td>frank</td><td>Remove</td></tr><tr><td>Add</td><td></td></tr></table>		User	Action	frank	Remove	Add	
User	Action							
frank	Remove							
Add								
Send only to	<input type="text" value="Email"/>							
Default message	<input checked="" type="checkbox"/>							
Conditions	<table><tr><th>Label</th><th>Name</th><th>Action</th></tr><tr><td colspan="3">No conditions defined.</td></tr></table>		Label	Name	Action	No conditions defined.		
Label	Name	Action						
No conditions defined.								
Operation condition	<table><tr><td><input type="text" value="Event acknowledged"/></td><td><input type="text" value="="/></td><td><input type="text" value="Not Ack"/></td></tr><tr><td>Add</td><td>Cancel</td><td></td></tr></table>		<input type="text" value="Event acknowledged"/>	<input type="text" value="="/>	<input type="text" value="Not Ack"/>	Add	Cancel	
<input type="text" value="Event acknowledged"/>	<input type="text" value="="/>	<input type="text" value="Not Ack"/>						
Add	Cancel							
Add Cancel								

图4-9

至此, 报警动作就已经配置好了。

4.7 收到第一封报警邮件

在前面的章节，我们依次从添加服务器到添加 Item，再到添加 Trigger 和报警动作进行了讲解，相信读者朋友们已经了解 Zabbix 的基本操作和一些基本概念了。

经过前面的配置，并且由于我们设置的报警条件是 CPU Load 大于 0，这时应该可以收到报警了。下面是收到的报警邮件内容示例。

```
Trigger: CPU Load is too high
Trigger status: PROBLEM
Trigger severity: High
Trigger URL:
Item values:
1. CPU Load(Zabbix server:system.cpu.load):0.32
2. *UNKNOWN*(*UNKNOWN*:*UNKNOWN*):*UNKNOWN*
3. *UNKNOWN*(*UNKNOWN*:*UNKNOWN*):*UNKNOWN*
Original event ID: 3158627
```

这是 Zabbix 默认的报警内容，里面写了 Trigger 的相关信息，以及与之关联的 Item 的值。作为最简单的使用来说，这些信息足以让我们了解到出问题的原因了。

4.8 Zabbix 报警流程

至此，我们已经学习了将服务器加入监控，添加 Item，添加 Trigger 和报警动作，最后收到报警邮件。其中穿插了新建用户和 Zabbix 事件等概念。下面给出一个直观的 Zabbix 的报警流程，希望能帮助读者清晰地理解 Zabbix 的核心概念。

首先 Item 收集数据，收集到数据后如果它和一个 Trigger 绑定了，那么会检查 Trigger 是否变成“异常”状态，然后生成一个事件（无论 Trigger 状态变化与否），最后会检查报警动作，如图 4-10 所示。

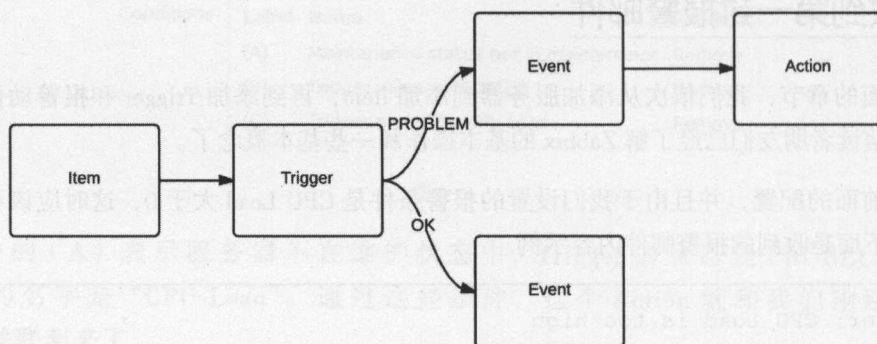


图4-10

4.9 看，Zabbix在工作呢

前文在安装 Zabbix 的时候，全都是一片黑色的命令行，大家不要担心，在使用 Zabbix 的过程中，大部分时间都是使用 Zabbix 前端。虽然说 Zabbix 前端不是很美观，但功能很强大（结果就是数据量大了以后会卡）。

首页屏幕上方是个两级菜单栏。第一行是最主要的几个分类：“Monitoring（监控）”、“Inventory（设备）”、“Reports（报表）”、“Configuration（配置）”、“Administration（管理）”。第二行菜单会根据第一行的选择进行变换。第三行是提示现在所在的位置，并可以跳转至之前的任何一层，如图 4-11 所示。



图4-11

搭建 Zabbix 后，进入“Reports”中的“Status of Zabbix”，如图 4-12 所示。这个页面是显示 Zabbix 目前的状态的，包括 Zabbix 目前监控了多少台 Host、多少个 Item 和多少个 Trigger。最后一行，有一个重要指标 VPS（value per second），意为每秒的监控数据。这个数字是通过监控的 Item 数量计算出来的，具体的计算方式，在后文会提到，它可以比较直观地衡量现在 Zabbix 承受的压力有多少。

Monitoring Inventory Reports Configuration Administration		
Status of Zabbix	Availability report	Triggers top 100 Bar reports
History: Configuration of discovery rules » Configuration of media types » Configuration of actions » Configuration of media types » Dashboard		
STATUS OF ZABBIX		
Parameter	Value	Details
Zabbix server is running	No	localhost:10051
Number of hosts (monitored/not monitored/templates)	39	1 / 0 / 38
Number of items (monitored/disabled/not supported)	74	66 / 0 / 8
Number of triggers (enabled/disabled) [problem/ok]	44	44 / 0 [0 / 44]
Number of users (online)	2	1
Required server performance, new values per second	1.01	-

图4-12

4.9.1 全局搜索框

在 Zabbix 前端的所有界面上，都有一个搜索框，可以让我们随时搜索，结果包括 Host、Host group 和 Template。

搜索框支持自动提示，如图 4-13 所示。

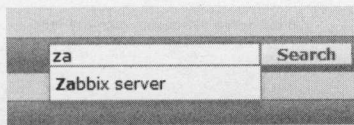


图4-13

大家不要惊奇为什么笔者要把这么简单的一个功能单独列出来。在 Zabbix 成熟后，上面会有很多的服务器，当你要快速定位到一个 Host 搜索框时，这是最简单有效的方法。

4.9.2 查看监控数据

监控了服务器后，会有很多监控数据，这些数据肯定是要拿来看的。这一小节主要介绍怎样在 Zabbix 前端查看需要的数据。

首先使用右上角的搜索框搜索到服务器，然后单击“Latest Data（最新数据）”，就可以看到 Item 了。单击某个“Application（后文会介绍，这里简单理解为 Item 的分类）”，就可以看到展开的 Item 了，Item 名字的右边就是目前 Item 的数据，旁边的加号和减号表示这次取到的 Item 数据相对于上一次的变化，如图 4-14 所示。

Name +	Last check	Last value	Change	
CPU (13 Items)				
Filesystems (10 Items)				
General (4 Items)				
log (1 Item)				
Memory (5 Items)				
Available memory	11 Apr 2014 09:07:36	5.44 GB	+232 KB	Graph
Free swap space	11 Apr 2014 09:07:29	9.75 GB	-	Graph
Free swap space in %	11 Apr 2014 09:07:30	100 %	-	Graph
Total memory	11 Apr 2014 08:28:37	7.8 GB	-	Graph
Total swap space	11 Apr 2014 08:28:31	9.75 GB	-	Graph
Network interfaces (2 Items)				
OS (7 Items)				
Performance (13 Items)				
Processes (2 Items)				

图4-14

单击旁边的“Graph”，可以看到这个 Item 的历史数据的趋势，如图 4-15 所示。

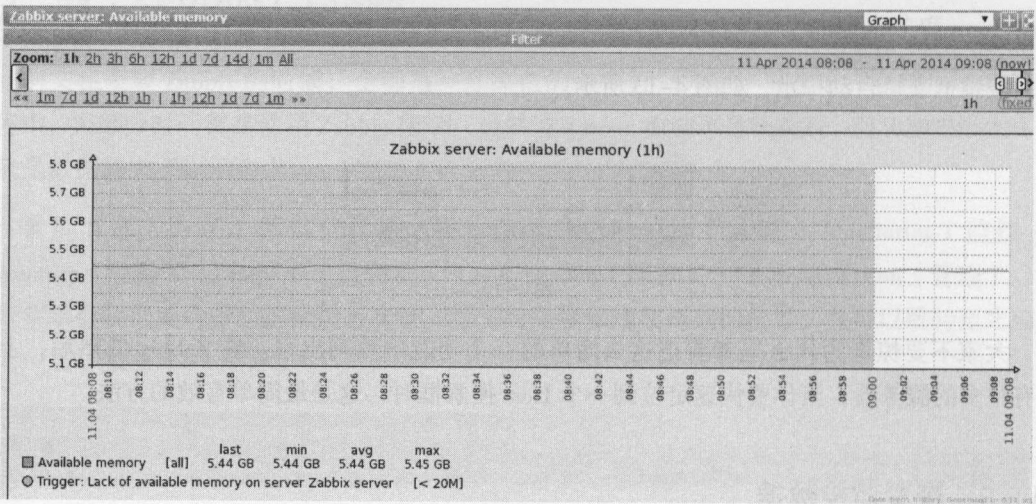


图4-15

4.9.3 查看报警信息

收到了一封报警邮件后，要怎么去查看这个报警呢？

在页面上进入“Monitoring”→“Events”，可以浏览最近的报警信息，如图 4-16 所示。

Time	Description	Status	Severity	Duration	Actions
01 Apr 2014 08:20:34	app.log is error	OK	Not classified	1d 17h 56m	-
31 Mar 2014 23:58:34	app.log is error	PROBLEM	Not classified	8h 22m	Failed
31 Mar 2014 23:58:34	app.log is error	PROBLEM	Not classified	0	Failed
31 Mar 2014 23:58:34	app.log is error	PROBLEM	Not classified	0	Failed
30 Mar 2014 16:00:07	Zabbix agent on Zabbix server is unreachable for 5 minutes	PROBLEM	Average	3d 10h 16m	Failed

图4-16

在此还可以看到 Zabbix 捕获到的异常，不单单显示出错的报警，当一个问题恢复时，也会有显示。

除了在 Events 中，还可以在“Monitoring”→“Triggers”中查看某个 Trigger 的历史状态，如图 4-17 所示。

Severity	Status	Info	Last change	Age	Host	Name ↑	Comments
Average	PROBLEM		30 Mar 2014 16:00:07	3d 10h 18m	Zabbix server	Zabbix agent on Zabbix server is unreachable for 5 minutes	Add
Average	PROBLEM		22 Jan 2014 22:10:35	2m 10d 4h	Zabbix server	Zabbix discoverer processes more than 75% busy	Add

图4-17

4.10 添加自定义监控点

Zabbix 自带了很多监控点，但总会有一些需求是 Zabbix 不支持的。在这种情况下，可以添加自定义的监控点。

对于自定义监控点，只要提供一个可以输出值的脚本即可。假设有个名为 mysql.sh 的脚本，用于返回本机运行的 MySQL 是否正常，返回值是 0 和 1。我们要做的，就是在前端配置 Item 的时候，使用 key 来和后端的脚本关联起来。

当脚本在 /tmp 下时，需要在“zabbix_agentd.conf”的“UserParameter”中写如下配置。

```
UserParameter=test_mysql, sh /tmp/mysql.sh
```

重启 Zabbix Agent 后，进行如下测试。

```
[apps@vlp-developer-201-234 zabbix]$ zabbix_get -s localhost -k test_mysql
1
```

如果 mysql.sh 需要传入参数，比如，不仅要能监控本机的 MySQL，还要能监控远程服务

器的 MySQL，需要将 IP 地址传入。这时格式是这样的：UserParameter=key[*],command，其中“*”的意思是方括号中可以有任意多的参数，里面的每个参数由逗号分隔，分别是 \$1, \$2……，它们表示脚本命令行接收到的参数。若需要方括号里的参数作为 command 中的变量名，比如 key[1]，要根据“1”来运行 awk '{print \$1}'，这时 command 中写 awk '{print \$1}' 会有问题，应该写 awk '{print \$\$1}'。下面看几个例子。

(1) UserParameter=ping[*],echo \$1：打印方括号中的第一个变量，比如 ping[0] 会返回“0”。

(2) UserParameter=mysql.ping[*],mysqladmin -u\$1 -p\$2 ping | grep -c alive：将 mysqladmin 的用户名密码传递到后端。

(3) UserParameter=wc[*],grep -c “\$2” \$1：grep 行数。

所以要这样配置：

```
UserParameter=test_mysql[*], sh /tmp/mysql.sh $1
```

这样的好处是不需要在前面（即上面例子中的 test-mysql）的参数列表中写非常长的参数。

如果定义的 UserParameter 太多，那么配置文件将很难阅读，Zabbix 支持将这些配置写在文件中，然后在 zabbix_agentd.conf 中设置配置文件的路径。比如把“UserParameter=test_mysql[*], sh /tmp/mysql.sh \$1”写在 zabbix_mysql.conf 中，然后在 zabbix_agentd.conf 中的“Include”中配置 zabbix_mysql.conf 或者它所在的目录即可。

第 2 章

增加监控



第二部分 配置篇

- ★ 第 5 章 增加监控
- ★ 第 6 章 报警配置
- ★ 第 7 章 数据可视化
- ★ 第 8 章 Users 和 Macros
- ★ 第 9 章 IT services 服务监控与 Web monitoring 网络监控
- ★ 第 10 章 Zabbix 前端界面
- ★ 第 11 章 Discovery

第 5 章

增加监控

前文用一个最简单的例子说明了 Zabbix 的基本配置流程，即 Zabbix 有 Server（服务端）和 Agent（客户端），Server 会根据 Item（监控项）的 Interval 从 Agent 获取数据（也有其他方式，但这是最简单的）。

当 Item 收到新的数据时，如果有 Trigger（触发器）与之关联，那么 Zabbix 就会根据 Item 的值检查这个 Trigger，得出的结果会生成一个 Event（事件），如果有符合要求的 Action（报警动作），那么就要进行 Action 中定义的操作。

在这一部分，主要讲解 Zabbix 各个功能的具体配置，希望能帮助读者把 Zabbix 提供的功能都理解清楚。这一章的各个小节之间基本没有依赖关系，可以选择自己最需要的部分进行阅读。比如目前工作在给服务器添加 Item 阶段的读者，可以着重阅读 Item 部分。在 Zabbix 的模型中，Host 是最小的物理实体，最常见的是服务器，也可以是交换机等网络设备。Host 上面会有 Item、Trigger 和 Graph，会有两种关联情况：一种是 Item 先和 Template 关联，再和 Host 关联；另一种是 Item 直接和 Host 关联。关于 Template 的内容后文会有详细介绍，这里只要知道它是一组 Item、Graph、Trigger 的集合就可以了。

进入 Host 的详细信息，可以看到，如果是 Template 上的 Item，那么 Item 前面会显示 Template 的名字，如图 5-1 所示。

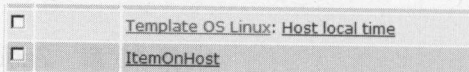


图 5-1

Host group, 是指一些 Host 的分组, 比如可以将一个机房的机器分为一个 Host Group, 也可以将服务于同一个业务的 Host 分为一个 Host group。

5.1 Host配置

从菜单中进入“Configuration”→“Hosts”, 任意点开一个 Host, 就能看到下面这些配置信息。

1. Host 标签页

Host 标签页如图 5-2 所示。下面进行详细分析。

The screenshot shows the Zabbix Host configuration interface. At the top, there are input fields for 'Host name' (containing 'Zabbix server') and 'Visible name'. Below these are two list boxes for 'Groups': 'In groups' (containing 'Zabbix servers') and 'Other groups' (containing 'Discovered hosts', 'Hypervisors', 'Linux servers', 'Templates', and 'Virtual machines'). There are left and right arrow buttons between these lists. Below the groups is a 'New group' input field. The 'Agent interfaces' section contains a table with columns: IP address (127.0.0.1), DNS name, Connect to (IP), Port (10050), and Default (checked). There are 'Add' and 'Remove' buttons for this section. Below are sections for 'SNMP interfaces', 'JMX interfaces', and 'IPMI interfaces', each with an 'Add' button. The 'Monitored by proxy' section has a dropdown menu set to '(no proxy)'. The 'Status' section has a dropdown menu set to 'Monitored'. At the bottom are buttons for 'Save', 'Clone', 'Full clone', 'Delete', and 'Cancel'.

图5-2

- Host name : 这是在 Zabbix 唯一的 Host 属性标识, 允许字母、数字、空格和下画线。需要注意的是, 输入的 Host name, 必须和这台 Host 上的 zabbix_agent.conf 中的 Hostname 属性一致, 这是 active checks 所必须的。active check 中, Host 会把 Item 数据发送给 Server, Server 需要用 Hostname 来确定发送过来的是哪一个 Host 的数据。
- Visible name : 在 List 和地图等地方显示的名字, 支持 UTF-8 编码, 可以使用中文。
- Groups : 选择 Host 属于的 Host Group。注意, 一个 Host 至少要在一个 Host Group 中。
- New group : 新建一个 Host Group, 并且这个 Host 将属于这个新建的 Host Group。如果留空的话就不会新建。

◎ Agent Interfaces : Interface 是 Zabbix 2.X 的新概念，其功能是在 Host 上添加一个接口，类似 Java 中的接口。这里可以选择的有 Agent、SNMP、JMX 和 IPMI。注意，在 Items 中使用的 Interface 是不能被移除的，Remove 连接是灰色的。Interface 一共有下面几种。

- Agent Interface : Zabbix Agent 的监控接口。
- SNMP Interface : SNMP 协议的监控，使用它可以获取一些硬件设备的信息。
- JMX Interface : 简单来说 JMX 是监控 Java 时使用的。
- IPMI Interface : 一种协议，一般用来获取服务器上的硬件信息和远程管理。

对于每一个 interface，都有下面这些设置。

- IP address : Host 的 IP 地址（可选）。
 - DNS name : Host 的 DNS 名字（可选）。
 - Connect to : 这里可以选择 IP 或者 DNS。选择 IP 的话，会根据输入的 IP 地址去寻找 Host；选择 DNS 的话会根据输入的 Host name 和 DNS 地址去解析 IP。
 - Port : Zabbix 使用的 TCP 端口，默认使用 Host 的 10050。
 - Default : 默认使用这个 Interface 和 Zabbix Agent 通信，比如在服务器有多块网卡时。
- ◎ Monitored by proxy : Host 可以被 Zabbix Server 监控，也可以被 Zabbix Proxy 监控（后文会介绍）。这里可以选择被哪个 Proxy 监控。
- ◎ Status : Host 的状态。Monitored 就是需要监控；Not Monitored 则相反。有时只是加入一台 Host 而不监控。

下面是对于 Zabbix 支持的 SNMP、JMX、IPMI 的介绍和说明。

（1）SNMP

SNMP（Simple Network Management Protocol，简单网络管理协议）构成了互联网工程工作小组（IETF，Internet Engineering Task Force）定义的 Internet 协议簇的一部分。该协议能够支持网络管理系统，用以监测连接到网络上的设备是否有任何引起管理上关注的情况。它由一组网络管理的标准组成，包含一个应用层协议（Application Layer Protocol）、数据库模型（Database Schema）和一组数据对象。在典型的 SNMP 用法中，有许多系统被管理，而且有一或多个系统在管理它们。每一个被管理的系统上运行一个叫做代理者（Agent）的软件组件，且通过 SNMP 对管理系统报告信息。

基本上，SNMP 代理者以变量呈现管理数据。管理系统通过 GET、GETNEXT 和 GETBULK 协议指令取回信息，或是代理者在没有被询问的情况下，使用 TRAP 或 INFORM 传送数据。管

理系统也可以传送配置更新或控制的请求，通过 SET 协议指令达到主动管理系统的目的。配置和控制指令只有当网络基本结构需要改变的时候使用，而监控指令则通常是常态性的工作。

可通过 SNMP 访问的变量以层次结构的方式结合。这些分层和其他元数据（例如变量的类型和描述）以管理信息库（MIBs）的方式描述。

（2）JMX

JMX（Java Management Extensions，即 Java 管理扩展）是 Java 平台上为应用程序、设备、系统等植入管理功能的框架。JMX 可以跨越一系列异构操作系统平台、系统体系结构和网络传输协议，灵活的开发无缝集成的系统、网络和服务管理应用。

（3）IPMI

IPMI（Intelligent Platform Management Interface，智慧平台管理接口）能够横跨不同的操作系统、固件和硬件平台，可以智慧型地监视、控制和自动回报大量服务器的运作状况，以降低服务器系统成本。IPMI 独立于操作系统外自行运作，并允许管理者在缺少操作系统或系统管理软件，或受监控的系统关机但连接电源的情况下仍能远端管理系统。

2. Template 标签页

Linked templates	Name	Action
	Template App Zabbix Server	Unlink Unlink and clear
	Template OS Linux	Unlink Unlink and clear

Link new templates

[Add](#)

[Save](#)
[Clone](#)
[Full clone](#)
[Delete](#)
[Cancel](#)

图5-3

如图 5-3 所示，是将 Template 连接到 Host 或取消连接的地方。要将一个 Template 关联到 Host 非常简单，在“type here to search”输入需要的 Template，根据提示选择后单击“Add”即可。取消关联包括两种情况：“Unlink”和“Unlink and clear”，它们的区别如下。

- Unlink：移除 Template 和 Host 的关联关系，但不会移除在 Template 上的 Item 等与 Host 的关系。
- Unlink and Clear：除了“Unlink”外，在 Template 上的 Item 等也取消与 Host 的关联。

3. IPMI 标签页

主要是 IPMI 相关的配置，此处不展开赘述。

4. Macros 标签页

用于定义 Host 级别的 Macro，这是 Zabbix 非常棒的功能，第 8 章会有详细介绍。

5. Host inventory 标签页

Inventory 指的是 Host 的一些硬件信息，或是资产信息，比如地理位置、CPU 型号和系统版本等。可以以手动或自动的方式输入 Host 的 inventory，也可以选择什么都不填。当选择自动的时候，可以用 Item 的值作为 Inventory 的值，比如以下这些常用的情况。

- ◎ system.hw.chassis[full|type|vendor|model|serial]：默认是 [full]，需要 root 权限。
- ◎ system.hw.cpu[all|cpunum,full|maxfreq|vendor|model|curfreq]：默认是 [all,full]。
- ◎ system.hw.devices[pci|usb]：默认是 [pci]。
- ◎ system.hw.macaddr[interface,short|full]：默认是 [all,full]，interface 填的是正则匹配。
- ◎ system.sw.arch：系统架构，如 i386。
- ◎ system.sw.os[name|short|full]：默认是 [name]。
- ◎ system.sw.packages[package,manager,short|full]：默认是 [all,all,full]，package 填的是正则。

如果想把 Agent ping 这个 Item 的值自动作为 Inventory 中 Type 的值，可以按以下步骤操作。

- (1) 设置需要的 Item，并把 Populated host inventory field 这一栏设置为 Type。
- (2) 更新这个 Item 时，Host 的 Inventory 会把 Agent ping 的数据作为 Type 的值。

这里有一个问题需要注意：如果不想把 Agent ping 这个 Item 作为 Type 的值，而是要取消或者作为其他 Inventory 的值，那么 Type 这个 Inventory 的值会保留下来。如图 5-4 所示，“Up(1)”是 Agent ping 的值，一开始把它作为“Type”的值，然后改成“Alias”，再改成“Tag”，可以看到虽然已经把 Agent ping 关联到“Tag”了，但“Type”和“Alias”还是会保留这个值。笔者认为，这个关联关系，是将 Item 的值写到 Inventory 的 MySQL 表中，如果取消关联关系，就不会把 Item 新的值写到 Inventory 表，但已经在 Inventory 表中的数据，是不会被删除的。

```

Type Up (1)
Name vlp-developer-201-234
Alias Up (1)
OS Linux vlp-developer-201-234 2.6.18-308.el5 #1 SMP Tue Feb 21 20:
Tag Up (1)

```

图5-4

6. Host group

Host group 在 Zabbix 中是一个比较简单的组织结构,它的功能是将一批 Host 放在一个组中,使查询数据时更加便捷。在配置 Host group 方面也非常简单:从菜单进入“Configuration”→“Host group”,选择“Create”就可以新建一个“Host group”。在弹出的页面上,只需要指定 Group 的名字和需要放入这个 Group 的 Host 即可。

5.2 Item属性

Item 是 Zabbix 的核心,也是监控系统的直接监控对象。事实上,对 Host 的监控和报警都是针对 Host 上的 Item 进行的。Item 可配置的属性非常多,下面会详细说明。

- Host : Item 属于的 Host 或者是 Template, 这里是灰色的, 不能更改。
- Name : Item 的名字。名字中可以使用 \$1, \$2 等参数。\$1 指的是下面 Key 里面, 括号中的参数。比如 vfs.fs.size[/,free], 如果标题是用“\$1 空余空间”, 则标题显示的就是“/ 空余空间”。
- Type : 这个会比较复杂, 后面会用一个小节来单独讲解。
- Key : 告诉 Zabbix 监控内容的属性。对于一个 Host, 它所有的 Item 的 Key 都是独一无二的, 如果试图设置两个相同的 Key, 会失败。另外, 当选择 Type 为“Zabbix Agent”, “Zabbix Agent (active)”, “Simple check”或者“Zabbix aggregate”时, 输入的 Key 一定要是 Zabbix 支持的。
- Host interface : 修改直接关联在 Host 上的 Item 时有效。
- Type of information : Item 取值的类型, 有下面几种选择。
 - Numeric (unsigned) : 64bit 无符号整型。
 - Numeric (float) : 浮点型。

- Character : 字符串, 大小不超过 255bytes。
 - Log : log 文件, 当使用 log 或者 logrt 作为 item 的 key 的时候需要选择的选项。
 - Text : 文本, 没有长度限制。
- ◎ Units : 如果这个选项被设置了, 那么 Zabbix 会针对返回值进行单位转换的处理。对于超过 1000 的数值, Zabbix 会将它除以 1000, 并以多少 k 的方式显示。比如 15000, 显示的时候就是 15k。比较特殊的是针对 B (byte) 和 Bps (bytes per second), 如果设置了 B 或者 Bps, 那么 1 会显示为 1B/1Bps, 1024 会显示为 1KB/1KBps。除了单位的转换, 还有时间的转换。当选择 unixtime 的时候, 返回值会被翻译为 “yyyy.mm.dd hh:mm:ss” 的样式, uptime 则会被翻译成 “hh:mm:ss” 或者 “N days, hh:mm:ss”。比如 Zabbix 接收到的是 881764 (秒), 那么就会显示为 “10 days,04:56:04”。此外, 如果选择 s, 那么显示的格式是 “yy mmm ddd hhh mmm sss ms”, 注意, 这里只会显示最大的三个单位, 比如 “1y 2m 3d”, “1h 2m 3s”。
- ◎ Use custom multiplier : 如果开启这个属性, 所有收到的整型或者浮点数都会乘以这里设置的数字。这主要是为了单位的转换, 比如把 KB, MBps 转换成 B, Bps。注意, 不能在前面加上前缀, 比如 K、M、G 等。从 Zabbix2.2 开始, 可以使用科学计数法。
- ◎ Update Interval (秒): 每 N 秒获取 Item 的数值。注意: 如果设置为 0, 那么数据不会刷新。如果 Flexible interval 被设置了非 0 的数值, 会以 Flexible interval 为准。
- ◎ Flexible intervals : 这个是一个弹性的 interval, 可以根据不同时间来设置不同的 Update Interval。比如可以设置为 10 点到 18 点的 Update Interval 是 10 秒, 19 点到 20 点的 Update Interval 是 20 秒。但这个对 Active 类型的 Item 是不生效的, 因为 Active 是 Agent 发出的数据。
- ◎ Keep trends (in days) : Zabbix 的历史数据有 History 和 Trends, Trends 保存了每小时的 min、max、avg、count 值。Zabbix 会保存 N 天的 Trends 的数据在数据库。更老的数据会被 Housekeeper 清除。从 Zabbix 2.2 开始, 这个值会被全局 Administration-General-Housekeeper 的设置覆盖。针对非数字的数据是不生效的, 因为非数字的数值不会保存 Trends。
- ◎ Store value : 这个是数据在获取后的一些操作, 具体如下。
- As is : 不做任何处理。
 - Delta (speed per second) : 这是计算一个差值 $(value - prev_value) / (time - prev_time)$, value 表示这次获取的数据, value_prev 是上次获取的数据, time 表示当前的时间戳, prev_time 是上一次获取数据的时间戳。这个设置非常有用, 特别是针对那种需要抓取的每分钟变化值的 Item。

注意：如果当前值比上一次的值要小，那么 Zabbix 不会记录这次变化（不存储任何东西），然后等待下一次值。

- Delta (simple change) : 就是 (value-prev_value)。

◎ Show value : 选择一种映射关系。数据返回“0”表示“服务器通电”，返回“1”表示“服务器负载高”，返回“2”表示“服务器磁盘空间不足”，等等。这个映射关系可以在“Administration”→“General”中配置，然后在下拉框中选择，Zabbix 已经自带了一些。

为了让 Zabbix 收集的监控数据更加可读，我们可以把数字的监控数据映射到文字的字符串。之前只能将 Numeric (unsigned) 类型的数据映射到字符串，从 Zabbix 2.2 开始，支持 Numeric (float) 和 Character 的映射。这个映射既在 Zabbix 前端生效，也在邮件 /SMS/Jabber 等 Media 中生效。

举个例子，一个 Item 的返回值是“0”和“1”，那么我们可以将其映射为：“0”——“Not Available”；“1”——“Available”，然后可以在 Item 配置的 Show value 选择事先定义的映射。

映射关系的设置，可以从菜单进入“Administration”——“General-Value mapping”，再选择 Create value map，在新建窗口中进行。映射结果的查看，即假设一个 Item 已经使用了前面说的“0”和“1”的映射关系，那么我们就在这个 Item 的 Latest Data 中看到“Not Available”或者是“Available”，如果没有设置过映射，那么这里显示的就是 0 或者是 1 了。

◎ Log time format : 只有当 type 是 Log 的时候才有效，支持的占位符有以下几种。

- y : 年 (0001 ~ 9999)
- M : 月 (01 ~ 12)
- d : 日 (01 ~ 31)
- h : 小时 (00 ~ 23)
- m : 分钟 (00 ~ 59)
- s : 秒 (00 ~ 59)

时间戳最左边的空格会被排除。比如一个日志为“23480:20100328:154718.045 Zabbix agent started. Zabbix 1.8.2 (revision 11211)。”我们要匹配这个日志可以使用“pppppp:yyyyMMdd:hhmmss”。这里的“pppppp:”只要和前面的 yMdhms 不同就可以了，具体是什么并没有关系。

- ◎ New application : 新建一个 Application。
- ◎ Populated host inventory field : 在这里可以选择一个在 Host 中设置的 Inventory 属性, 这样的话, 该项 Item 的值会自动推送给选择的 inventory field。
- ◎ Description : 对于该项 Item 的描述。
- ◎ Enabled : 是否启用该项 Item。

5.3 Item类型

5.3.1 Zabbix Agent类型

Zabbix agent 类型的 Item 是使用部署在服务器端的 Agent 来获取数据的, 可以分为被动模式和主动模式。

Item 的 Type 中, 有 “Zabbix agent” 和 “Zabbix agent (active)” 两个容易混淆的概念, 如图 5-5 所示。

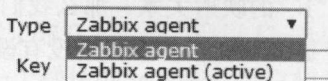


图5-5

一般把前者称为 “被动模式”, 后者称为 “主动模式”, 下面主要介绍这两者的功能和区别。

总的来说, Zabbix 和 Zabbix Agent 之间的数据传输协议是一种类似 JSON 的数据格式。Zabbix 使用的协议, 有以下预先定义的值。

(1) <HEADER> : ZBXD\x01, 一共 5 个字节。

(2) <DATALEN> : 这次连接传送数据的长度, 一共 8 个字节。比如长度为 “1”, 那么这里就是 “01/00/00/00/00/00/00/00”, 即十六进制中的 8 个字节, 64 位数字。

为了限制内存的消耗, 每一个连接传送的数据最大不能超过 64MB。

1. 被动模式

被动模式是最简单的, Zabbix Server 或者 Proxy 向 Zabbix Agent 请求数据, 然后 Zabbix Agent 将数据发送回 Zabbix Server 或者 Proxy。Zabbix Server 请求的内容为 :

```
<item key>\n
```

而 Zabbix Agent 的返回的格式为：

```
<HEADER><DATALEN><DATA>
```

当 Zabbix Server 请求 agent.ping 这个 key 的 Item 的时候，会有以下的步骤。

(1) Zabbix Server (或者 Proxy, 下同) 打开一个 TCP 连接。

(2) Zabbix Server 发送 “agent.ping\n”。

(3) Zabbix Agent 读取请求，并且返回 “<HEADER><DATALEN>1”，这里的 <HEADER> 和 <DATALEN> 在本节开头处的 Zabbix 传输协议中介绍过。

(4) Zabbix Server 获取 Zabbix Agent 返回的数值，即例子中的 “1”。

(5) TCP 连接关闭。

通过这个例子，相信大家已经对被动模式有所理解了。所谓的“被动”，是针对 Zabbix Agent 来说的，每一次 Item 值的获取，对于 Zabbix Agent 来说是“被动”的，因为是由 Zabbix Server 发起的请求。接下来我们看看主动模式。

2. 主动模式

主动模式和被动模式相反，被动模式是 Zabbix Server 发起请求，而主动模式是 Zabbix Agent 主动将数据发送给 Zabbix Server。在主动模式中，Zabbix Agent 会先从 Zabbix Server 获取需要发送的 Items 的列表，然后根据列表中的 Item，去获取数据，再发送给 Zabbix Server。这里 Zabbix Agent 请求的 Zabbix Server 的列表，是在 zabbix_agentd.conf 中的 “ServerActive” 参数中定义的，另一个参数 “RefreshActiveChecks” 定义了 Zabbix Agent 每隔多久向 Zabbix Server 请求一次。如果某一次请求失败了，那么会在 60 秒后重试。

首先 Zabbix agent 请求需要发送的 Item 列表，请求的内容如下。

```
<HEADER><DATALEN>{
  "request": "active checks",
  "host": "<hostname>"
}
```

这时 Zabbix Server 的返回内容如下。

```

{
  "response": "success",
  "data": [
    {
      "key": "log[\\home\\zabbix\\logs\\zabbix_agentd.log]",
      "delay": "30",
      "lastlogsize": "0"
    },
    {
      "key": "agent.version",
      "delay": "600"
    }
  ]
}

```

整个过程包括如下几步。

- (1) Zabbix Agent 打开一个 TCP 连接。
- (2) Zabbix Agent 请求需要检查的列表。
- (3) Zabbix Server 返回需要 Item 的列表。
- (4) Zabbix Agent 处理 Zabbix Server 的返回值。
- (5) 关闭 TCP 连接。
- (6) Zabbix Agent 开始定期收集数据。

当 Zabbix Agent 将数据发送给 Zabbix Server 的时候，格式如下。

```

<HEADER><DATALEN>{
  "request": "agent data",
  "data": [
    {
      "host": "<hostname>",
      "key": "log[\\home\\zabbix\\logs\\zabbix_agentd.log]",
      "value": "13039:20090907:184546.759 zabbix_agentd started. ZABBIX 1.6.6
(revision {7836}).",

```

```

"lastlogsize":80,
"clock":1252926015
},
{
"host": "<hostname>",
"key": "agent.version",
"value": "1.6.6",
"clock":1252926015
},
{
"clock":1252926016
}

```

然后 Zabbix Server 的返回内容如下。

```

<HEADER><DATALEN>{
"response": "success",
"info": "Processed 2 Failed 0 Total 2 Seconds spent 0.002070"
}

```

注意：如果 Zabbix Agent 发送的一些数据在 Zabbix Server 上出错了，比如 Host 和 Item 被 disabled 了或者根本不存在，那么 Zabbix Agent 是不会重试的。

对于 Zabbix Agent 发送这个过程，具体步骤如下。

- (1) Zabbix Agent 打开一个 TCP 连接。
- (2) Zabbix Agent 发送数据的列表。
- (3) Zabbix Server 获取到数据列表，并进行处理。
- (4) TCP 连接关闭。

5.3.2 SNMP 类型

SNMP 是监控服务器以外设备的非常好的方式，比如可以用于监控打印机、交换机、路由器等，只要有 SNMP 功能的，Zabbix 都可以监控。在设置 SNMP Item 的时候，只要将 Type

选择为 SNMP 即可。具体步骤如下。

(1) 首先添加 SNMP 设备到 Zabbix。

(2) 然后获取设备支持的所有 SNMP 信息。这里使用 `snmpwalk`（它可以在安装 Zabbix 时用 `net-snmp` 一起安装）。

```
>shell snmpwalk -v 2c -c public HOSTIP
```

其中“-v”参数表示 SNMP 版本,这里使用的是 SNMPv2,“public”是一般使用 SNMP 的字段,如果用这条命令无法显示出 SNMP 信息,可以查找资料,看看对于你的设备,“public”应该填什么。如果需要获取交换机上端口 3 进来的网络流量,应该使用 IF-MIB::ifInOctets.3（这是之前的命令获取的）。

(3) 接着使用 `snmpget` 获取数据。

```
>shell snmpget -v 2c -c public -On HOSTIP IF-MIB::ifInOctets.3
```

可以获取下面这串结果。

```
.1.3.6.1.2.2.1.10.3 = Counter32:3472126941
```

(4) 在 Item 配置页面中, type 选择 SNMPv*, 星号表示 SNMP 版本, 最后在 OID 中输入刚获取的 .1.3.6.1.2.2.1.10.3。

注意：如果要使用 SNMP 监控, 必须在编译 Zabbix 的时候加上 `--with-net-snmp`。

5.3.3 IPMI类型

Zabbix 也有监控 IPMI 设备的需求, 在编译安装 Zabbix Server 的时候, 一定要加上 IPMI 选项 `--with-openipmi`。一般我们使用 IPMI 来监控硬件信息, 比如温度之类。默认 Zabbix Server 不会有 poller 来监控 IPMI, 这时候就算有 IPMI 的 Item, 也不会有任何数据, 需要在 `zabbix_server.conf` 中将 `StartIPMIPollers` 改成非 0 的数字。

具体步骤如下。

(1) 在将 Host 添加监控时, 就要加入 IPMI interface。

(2) 在 Item 的配置中选择 IPMI, 配置 IPMI sensor, 比如在 Dell Poweredge 上的“FAN MOD 1A RPM”

(3) 在 key 中输入 Host 独一无二的 Item key。

(4) 选择合适的数据类型和单位。

5.3.4 日志文件监控

下面介绍 Zabbix 另一个“重量级”的功能——日志文件的监控。相对于其他的日志工具，Zabbix 的日志监控工具很简单，但却很有效，它最主要的是监控日志文件中有没有某个字符串的表达式。对于日志轮转与否，Zabbix 都支持。

在配置 Item 的时候，Type 选择 Zabbix agent (active)，这里主要需要配置的是 Key。下面是监控日志的两种 key——log 和 logtr。

```
log[/path/to/some/file,<regexp>,<encoding>,<maxlines>,<mode>,<output>]
```

```
logtr[/path/to/some/filename_format,<regexp>,<encoding>,<maxlines>,<mode>,<output>]
```

只要配置了 <regexp>，Zabbix 会根据 <regexp> 的正则表达式来匹配日志中的内容。注意，一定要保证 Zabbix 用户对日志文件有可读权限，否则，这个 Item 的状态会变成“unsupported”。

需要注意以下几点。

(1) Zabbix Server 和 Zabbix Agent 会追踪日志文件的大小和最后修改时间，并且分别记录在字节计数器和最新时间计数器中。

(2) Agent 会从上次阅读日志的地方开始读取日志。

(3) 字节计数器和最新时间计数器的数据会被记录在 Zabbix 数据库，并且发送给 Agent，这样能够保证 Agent 从上次停止的地方开始读取。

(4) 当日志文件大小小于字节计数器中的数字时，字节计数器会变为 0，从头开始读取文件。

(5) 所有符合配置的文件，都会被监控。

(6) 一个目录下的多个文件如果修改时间相同，会按照字母顺序来读取。

(7) 到每个 Update interval 的时间时，Agent 会检查一次目录下的文件。

(8) Zabbix Agent 每秒发送的日志量，有一个日志行数的上限，这是为了防止网络和 CPU 负载过高，这个数字是 zabbix_agent.conf 中的 MaxLinesPerSecond。

(9) 发送的日志量最多是 Agent 设置的 buffer size 的 50%，就算只有日志类监控（即没有其他类型的 Item），也只有 50%。同时，由于有 maxlines，故需要将 zabbix_agent.conf 中的 BufferSize 设置为 maxlines 的 2 倍。

(10) 当 Agent 没有 Log 类型的监控的时候，所有的 BufferSize 都会为其他类型 Item 服务。如果这时新建了日志文件类型的监控，会将 BufferSize 中最老的 50% 数据清空给日志文件 Item 使用。

(11) 在 logrt 中，正则表达式只对文件名有效，对文件目录无效。

5.3.5 计算型Item

计算类型的 Item，可以通过计算其他 Item 来获取数据。可能一台服务器有多块网卡，需要将 4 块网卡的出口流量全部相加，这时，计算型 Item 就可以派上用场了。计算型 Item 可以新建一个虚拟的 Item，它的数据是通过计算其他的 Item 来获取的，可以自己定义这个计算的表达式。对于计算型 Item，Zabbix 会像其他 Item 一样看待它，也能建立 Trigger，并将数据存储在数据库中。在 Item 配置时，将 type 选择为 Calculated。

在 Item 的配置中，可以书写自己的计算表达式。Zabbix 支持的计算表达式如下。

```
func (<key>|<hostname:key>,<parameter1>,<parameter2>,...)
```

其中，func 是 last、min、max、avg、count 等 Zabbix 在 Trigger 中支持的函数。key 和 hostname:key 是引用其他 Item 的值。需要注意的是，最好在 key 的外面加上双引号，这样能防止 Zabbix 解析错误。如果在双引号内还要使用双引号，可以加上“\”来转移。后面的 parameter，就是 func 需要的一些参数。下面看一些具体的表达式实例。

- ◎ 计算磁盘空余空间百分比： $100 \times \text{last}("vfs.fs.size[/,free]",0) / \text{last}("vfs.fs.size[/,total]",0)$
- ◎ 计算 Zabbix 在 10 分钟内平均处理的数据量： $\text{avg}("Zabbix Server:zabbix[wcache,values]",600)$
- ◎ 计算两个 func 的和： $\text{last}("net.if.in[eth0,bytes]",0) + \text{last}("net.if.out[eth0,bytes]",0)$
- ◎ 双引号中嵌套双引号的例子，这里是对于一个聚合类型的 Item 再做计算： $\text{last}("grpsum[\"video\\\", \"net.if.out[eth0,bytes]\", \"last\\\", \"0\\\"]\",0) / \text{last}("grpsum[\"video\\\", \"nginx_stat.sh[active]\", \"last\\\", \"0\\\"]\",0)$

注意：

1. 计算表达式中相关的 Item key 必须存在，而且有值，如果更改了和计算表达式相关联的 Item key，必须手动更改计算表达式中的 Item key。

2. 针对 User macros, 在计算表达式中会自动展开。但是, 如果 User macros 指代的是 func, host name, item key 或者是运算符, 它不会被展开。

5.3.6 Zabbix内部监控

Zabbix 内部监控是针对 Zabbix Server 或者 Zabbix Proxy 本身的一些指标的监控, 由 Server 或 Proxy 自己计算获得。Zabbix 内部监控也是由 Zabbix pollers 来完成的。

Zabbix 内部监控包括下列几项。

- ◉ zabbix[boottime] : Server 或者 Proxy 启动的时间, 值为 UNIX 时间戳。
- ◉ zabbix[history]、zabbix[history_log]、zabbix[history_str]、zabbix[history_text]、zabbix[history_unit] : HISTORY 表中存的数据量。如果是 MySQL InnoDB、Oracle 或者 PostgreSQL, 请不要使用它, 可能有性能问题, Proxy 不支持。(这是官方文档的说法, 笔者也不能确定, 各位还是宁可信其有不可信其无吧。)
- ◉ zabbix[host,<type>,available] : 检查某种类型在 Host 上是否可用。type 可以是 Agent、SNMP、IMPI、JMX。返回值“0”表示不可用,“1”表示可用,“2”表示未知。从 Zabbix 2.0.0 开始可用。Proxy 不支持。
- ◉ zabbix[hosts] : 监控 Host 或者 Items 的数量。
- ◉ zabbix[items] : 监控的 Items 数量 (包括可用或不可用)。
- ◉ zabbix[items_unsupported] : 不可用的 Items。
- ◉ zabbix[java,<param>] : 返回和 Zabbix Java gateway 相关的信息。当 <param> 是 ping 时, 会返回“1”, 可以配合 Trigger 中的 nodata() 来检查 Java gateway 是否可用。当 <param> 是 version, 会返回 Java gateway 的版本。注意: 第二个参数一定要留空, 为了以后使用。从 Zabbix 2.0.0 开始支持。
- ◉ zabbix[process,<type>,<mode>,<state>] : 查询 Server 或者 Proxy 在某个状态下所消耗的时间百分比。只会计算最近一分钟的数据。如果 <mode> 是一个不存在的进程号 (比如设置了 5 个 pollers, <mode> 中却填 6 个), Item 会变成“不可用”状态。self-monitoring 进程每分钟会去收集一次 Zabbix 进程在内存中进行的工作。busy 和 idle 只有状态变化时, Item 的值才会变化。这保证了一个挂起的进程也会被认为是 100%busy。在当前,“busy”表示进程不在“sleeping”状态, 在未来, Zabbix 会添加更多的状态, 比如等待锁、查询数据库等。在 Linux 上, 这个状态 0.01 秒变化一次。

其支持的进程如下。

- `alerter` : 发送通知的进程。Proxy 不支持。
- `configuration syncer` : 在内存中维护配置信息的进程。
- `data sender` : proxy 将数据发送到 server 的进程。Server 不支持。
- `discoverer` : 侦测设备的进程。
- `escalator` : 报警扩散的进程。Proxy 不支持。
- `heartbeat sender` : proxy 发送心跳的进程。Server 不支持。
- `history syncer` : Zabbix 将 history 写入数据库的进程。
- `housekeeper` : 删除旧的 history 数据的进程。
- `http poller` : web 监控的进程。
- `icmp pinger` : IMPI 检查的进程。
- `java poller` : Java 检查的进程。
- `node watcher` : node 之间发送配置信息更新和历史数据的进程。Proxy 不支持。
- `poller` : 进行被动检查的进程。
- `proxy poller` : 被动模式的 proxy 的 poller 进程。
- `self-monitoring` : 收集 Zabbix 自身数据的进程。
- `timer` : 处理时间相关的 Trigger 和 Maintenances 的进程。Proxy 不支持。
- `trapper` : 检查 active checks, traps, inter-node 和 proxy 通信的进程。
- `unreachable poller` : 检查设备是否和 Zabbix 网络互通的进程。
- `vmware collector` : 收集 VMware 数据的进程。

其支持的 mode 有 : avg、count、max、min。

其支持的 state 有 : busy (默认)、idle。

下面看几个例子。

- `zabbix[process,poller,avg,busy]` : 监控的是最近 1 分钟内 poller 进程平均工作时间。
- `zabbix[process,"icmp pinger",max,busy]` : 在最近 1 分钟内 icmp pinger 使用的最长时间。
- `zabbix[process,trapper,count]` : 目前运行的 trapper 进程数量。
- ◎ `zabbix[proxy,<name>,<praram>]` : 获取 Proxy 相关数据。<name> 是 Proxy 的名字, 对于 <praram>, 目前仅仅支持 lastaccess, 最后一次收到 Proxy 的心跳。
- ◎ `zabbix[proxy_history]` : Proxy 的 history 表中等待发送给 Server 的数据数量。Server 不支持。

- zabbix[queue,<from>,<to>]: 在 Zabbix 中延迟了 <from> 到 <to> 秒的数据。默认 <from> 是 6 秒, <to> 是无限, 所以默认就是超过 6 秒延迟的 Item 的数量。这里的 <from> 和 <to> 支持 smhdw, 即 Zabbix 自身定义的时间格式。
- zabbix[rccache,<cache>,<mode>]: Zabbix 配置使用的缓存情况。<cache> 只能使用 buffer, <mode> 可以使用以下各项。
 - total: buffer 的总大小。
 - free: buffer 的空闲大小。
 - pfree: buffer 空闲大小百分比。
 - used: buffer 已经使用的大小。
- zabbix[requiredperformance]: value per second, 即 Reports-Status of Zabbix 中的 vps。
- zabbix[trends|trends_unit]: 在 Trends 表中的数据数量, 在 InnoDB, Oracle 和 PostgreSQL 不要使用。Proxy 不支持。
- zabbix[triggers]: Zabbix 数据库中可用的 Triggers。Proxy 不支持。
- zabbix[uptime]: Zabbix Server 或者 Proxy 启动到现在的时间, 单位是秒。
- zabbix[vccache,buffer,<mode>]: Zabbix 监控数据缓存的使用状况, <mode> 可以是以下各项。
 - total: buffer 的总大小。
 - free: buffer 的空闲大小。
 - pfree: buffer 空闲大小百分比。
 - used: buffer 已经使用的大小。
- zabbix[vccache,cache,<parameter>]: Zabbix 监控数据缓存的性能指标, <parameter> 可以是以下各项。
 - requests: 总请求数。
 - hits: 缓存命中数 (history 数据会从缓存获取)。
 - misses: 缓存失败数 (history 数据从数据库中获取)。
- zabbix[vmware,buffer,<mode>]: vmware 的缓存内存使用状况如下。
 - total: buffer 的总大小。
 - free: buffer 的空闲大小。

- pfree : buffer 空闲大小百分比。
- used : buffer 已经使用的大小。

◎ zabbix[wcache,<cache>,<mode>] : Zabbix 写缓存的指标。当 <cache> 是 values 时, <mode> 可以是 all/float/unit/str/log/text/not supported, 即 Server 或者 Proxy 所有处理的不同类型的数据。当 <cache> 是 trend 或 text 时, <mode> 可以是 pfree、free、total、used, 含义和前面的相同。

5.3.7 ssh类型Item

ssh 检查是不需要在客户端服务器上安装 Zabbix Agent 的, 如果要使用 ssh 检查, 需要在编译 Zabbix Server 时加上 ssh2 的支持, 最低需要 libssh2 1.0.0 版本。

Zabbix 在安装完后默认在配置文件中没有开启 ssh 检查, 在 zabbix_server.conf 中, 将 SSHKeyLocation 改成运行 Zabbix Server 身份 home 下的 .ssh, 比如是使用 Zabbix 用户运行的, 那么就令 SSHKeyLocation=/home/zabbix/.ssh。/home/zabbix/.ssh 是默认的 ssh-gen, 是在 Linux 保存公钥和私钥的地方, 当然也可以放在别的地方。这一小节假设公钥和私钥放在 /home/zabbix/.ssh/ 中。

这里简单介绍一下公钥和私钥。传统的加密解密中, 比如要加密一段文本, 发给 $\times\times\times$, 如果在传送途中, 加密算法泄露了, 那么要传送的文本也就泄露了, 这个叫对称加密算法, 即加密解密用的是同一套算法。用数学表达式来看, 加密算法是 f , 文本是 str , 那么发送的内容是 $f(str)$, 接收的 $\times\times\times$ 想获取 str , 就要 $f^{-1}(str)$ 就行了。为了解决这种问题, 公钥和私钥是一种非对称加密, 即加密的算法和解密的算法是不一样的, 它们数学相关, 但无法通过一个计算出另一个。加密密钥和解密密钥中, 公开的是公钥, 不公开的是私钥。当加密密钥公开时, 是给私钥持有者发送数据加密数据用; 当公开的是私钥时, 用于验证消息是否是原作者发布的。

下面, 先在 Zabbix Server 上生成公钥和私钥。

```
# sudo -u zabbix ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/zabbix/.ssh/id_rsa):
Created directory '/home/zabbix/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```


Your identification has been saved in /home/zabbix/.ssh/id_rsa.

Your public key has been saved in /home/zabbix/.ssh/id_rsa.pub.

The key fingerprint is:

90:af:e4:c7:e3:f0:2e:5a:8d:ab:48:a2:0c:92:30:b9 zabbix@it0

The key's randomart image is:

+--[RSA 2048]-----+

```
|
|      .      |
|      o      |
|  .    o      |
|+    . S      |
|. +   o =      |
|E .   * =      |
|=o . . . * .   |
|... oo.o+      |
+-----+
```

这时在 /home/zabbix/.ssh 中就会有 id_rsa.pub 和 id_rsa 生成，前者是公钥，后者是私钥，由于使用的是 rsa 算法，所以名字中带有 rsa。然后将公钥复制到需要进行 ssh 监控的服务器上，cat 'id_rsa.pub' >> /home/zabbix/.ssh/authorized_keys。最后测试，从 Server 所在的服务器使用 Zabbix 用户 ssh 到这台 Host，如果能够不用密码顺利登录，那说明配置正确了。

Item 上的配置如下。

- Key : ssh.run[<唯一标识>,<ip>,<port>,<encoding>]。
- Authentication method : 可以选择密码或者是公钥。
- user name : 登录 Host 使用的用户名，必填。
- public key file : 如果之前选择公钥验证，这里要填写之前生成的公钥文件名，比如 id_rsa.pub。
- private key file : 如果之前选择私钥验证，填写之前创建的私钥文件，比如 id_rsa。
- password or key passphrase : 密码短语，是创建密码的一种方式。
- executed script : 远程执行的命令。

5.3.8 Telnet类型Item

Telnet 类型监控不需要在目标服务器上安装 Zabbix Agent。这个监控相当于 Telnet 目标 IP，输入一些命令，可以获取它的返回值。对于多个命令，可以分成几行来写，返回值也会被分成多行。\$、#、>、% 可以作为一行的结尾，使用这四个字符作为一行的结尾时，返回值将不包含这一行的结果。

具体的配置是：telnet.run[<unique short description>,<ip>,<port>,<encoding>]。

5.3.9 External Check类型Item

External 类型的检查是在 Zabbix Server 上运行脚本获取数据的监控。它不需要在 Zabbix Server 上安装 Zabbix Agent。它的配置是 script[<parameter1>,<parameter2>,...]，其中 script 是脚本名称，方括号中的是执行脚本的参数。

Zabbix Server 会到 zabbix_server.conf 中配置的目录下寻找脚本去执行，身份就是 Zabbix Server 运行的 Linux 用户，所以环境变量最好在脚本中设置，而且需要保证运行 Zabbix Server 的身份可以运行这些脚本。External 监控会将 stdout 的结果去除首尾空格后返回，忽略 stderr 的输出。

一个脚本 check_oracle.sh，它需要一个“-h”的参数指向 oracle 的 IP。那么将它放入 zabbix_server.conf 设置的脚本目录后，External 类型监控值的 Key 只要写 check_oracle.sh[“-h”，“1.2.3.4”] 就可以了，也可以用宏，写成 check_oracle.sh[“-h”，{HOST.CONN}]，这就会在 Zabbix Server 上运行 check_oracle.sh“-h”“1.2.3.4”。这里要注意两点。

(1) 不要过度使用 External 类型的 Item，它会使 Zabbix 系统的性能变差。

(2) 方括号中的参数，最好加上双引号，以免解析出错。

5.3.10 Aggregate类型Item

使用 Aggregate 类型的 Item，Zabbix 可以直接查询数据库，将一些数据聚合起来作为一个独立的 Item。由于它是直接查询数据库的，所以 Aggregate 类型的 Item 不需要在服务器上安装 Zabbix Agent，Aggregate 类型 Item 的 Key 的格式如下：

groupfunc[“Host group”，“Item key”，itemfunc,timeperiod]

下面进行具体分析。

Host group 表示要聚合哪些 Host 的数据, 从 Zabbix 1.8.2 开始, 可以支持多个 Host group, 它们之间使用逗号分隔。

groupfunc 指的是针对 Host group 中所有 Host 的 item 使用什么函数聚合, 目前支持的有:

- grpavg: 平均值。
- grpmax: 最大值。
- grpmin: 最小值。
- grpsum: 求和。

itemfun 指的是一段时期内 (这里需要使用 Zabbix 中定义时间的语法, 1d 表示 1 天, 5m 表示 5 分钟。如果没有写单位, 那么默认单位是秒)。针对 Item 的计算如下。

- avg: 平均值。
- count: 监控数据的个数。
- last: 最近一次监控值。
- max: 最大值。
- min: 最小值。
- sum: 求和。

注意:

- (1) 当 itemfunc 是 last 的时候, tiemperiod 这个参数将被忽略。
- (2) # 是不支持的。
- (3) 只有在被监控的 (enabled) 服务器上的活跃的 (active) 的 Item 才会被计算。

下面来看例子:

(1) grpsum["MySQL Servers","vfs.fs.size[/,total]",last,0], 表示 "MySQL Servers" 中 Host group 的总磁盘空间。

(2) grpavg["MySQL Servers","system.cpu.load[avg1]",last,0], 表示 "MySQL Servers" 所有服务器的平均 CPU 负载。

(3) grpavg["MySQL Servers",mysql.qps,avg,5m], 表示 "MySQL Servers" Host group 中的 5 分钟 mysql.qps 的平均值。

（4）`grpavg[["Servers A","Servers B","Servers C"],system.cpu.load,last,0]`，表示多个 Host group 服务器的平均 CPU 负载。

5.3.11 Trapper类型Item

Trapper 类型的 Item 和普通的 Item 相比，最大的不同在于 Trapper 类型的 Item 不依靠 Zabbix 主动向 Agent 查询数据，而是由 Agetnet 将数据推送给 Zabbix。如果要使用它，只需在 Item 中配置 Trapper 类型，然后将数据发送给 Zabbix 即可。

需要配置以下三个属性。

- ◎ Key：即 Item key，在发送数据时需要这个字段。
- ◎ Type of information：选择收到的数据的类型。
- ◎ Allowed Hosts：接受从哪些服务器发送过来的数据。如果有多个，用逗号分隔。从 Zabbix 2.2 开始，空格和用户宏都可以在这里使用。

在设置完发送数据前，最好等 60 秒，因为 Zabbix 会将配置缓存在内存中。

接下来我们试试发送数据，这里使用到了 `zabbix_sender`。

```
zabbix_sender -z <server IP address> -p 10051 -s "New host" -k trap -o  
"test value"
```

具体分析如下。

- ◎ `-z`：Zabbix Server IP。
- ◎ `-p`：Zabbix Server 的端口，默认是 10051。
- ◎ `-s`：发送数据的 Hostname（使用 technical 的 Hostname 即服务器自身的 Hsotname，而不是 visible 的 Hostname 即 Zabbix 显示的 Hostname。）
- ◎ `-k`：item key。
- ◎ `-o`：发送的具体数据。

5.3.12 JMX类型Item

JMX 的全称是 Java Management Extensions，即 Java 管理扩展。Java 程序会开放一些端口，用来获取运行状况。在 Zabbix 1.8 以前，只能使用 Zapcat 来监控 JMX，并需要修改源代码来支持，

非常麻烦。另一种方法是使用 `jmx-cmd-client`，它的作用是从命令行去获取 JMX 信息，可以在它的上层包装一个程序，用来获取 JMX 数据。

从 Zabbix 2.0 开始，内置了监控 JMX 的功能，叫做“Zabbix Java Gateway”，在 Zabbix Server 和 Zabbix Proxy 上会启动名为“Zabbix Java Gateway”的进程，当需要获取 JMX 数据时，Zabbix Server 会“问”JMX Gateway，然后 JMX Gateway 根据 JMX 管理 API 去查询需要的数据。在使用时，Java 程序不需要在代码中新增任何东西，只需要在启动时加上一些 JVM 参数，使得它可以支持使用端口监控 JMX。

比如可以进行如下设置。

```
java \
-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=12345 \
-Dcom.sun.management.jmxremote.authenticate=false \
-Dcom.sun.management.jmxremote.ssl=false \
-jar /usr/share/doc/openjdk-6-jre-headless/demo/jfc/Notepad/Notepad.jar
```

它启动了一个本地非常简单的 Java 程序，使用 12345 端口，具体的 jar 依情况各自不同，`ssl=false` 说明它不需要身份验证。JMX 设置身份验证的方法，这里不做详细说明，可以查询相应的 JMX 文档。

在配置时，要在需要进行 JMX 监控的服务器上添加 JMX interface，如图 5-6 所示。



图5-6

JMX Item Key 的其他一些细节如下。

(1) 简单的属性，即 MBean 的对象名称就是一个定义的字符串，它的返回值就是一个常规的类型，比如整形和字符串。对这类型的 JMX，Item Key 非常简单，就是 `jmx[com.example:Type=Hello,weight]`。

(2) 复杂的返回值：当返回一个 `HashMap` 的时候，使用点来分隔对象名称和 Key。比如 `jmx[com.example:Type=Hello,apple.weight]`，对象名字是 `apple`，我们需要其中的 `weight` 参数。同样的，如果返回的数据更加复杂，`HashMap` 里套 `HashMap`，可以用多个点号来处理，比如 `jmx[com.example:Type=Hello,fruits.apple.weight]`。

注意：

(1) 使用点号，但它不是当做对象名和属性名的分隔符的时候，需要加上反斜杠“\”进行转义。

(2) 如果有空格或者逗号，必须放在双引号中。

JMX 监控是 Zabbix 新加入的非常强大的功能，以前，只能使用 `cmdline-jmxclient` 来监控 JMX 信息。在命令行中使用的与下面类似命令：

```
java -jar cmdline-jmxclient-X.X.jar - localhost:8081 com.example:Type=Hello,fruits.apple.weight
```

在配置监控的时候比较痛苦，而使用了 Zabbix 自带的 JMX 监控后，就非常简单了。

5.3.13 ODBC类型Item

ODBC 监控就是 Zabbix 前端显示的“数据库类型监控”。ODBC 是用 C 语言写的和数据库管理系统（DBMS）打交道的中间层 API。ODBC 的概念是由微软创造的，之后推广到其他平台。

Zabbix 可以从任何支持 ODBC 的数据库获取信息。Zabbix 不会直接和数据库打交道，它通过 ODBC 接口和 ODBC 驱动来完成。Zabbix 使用 `unixODBC`，它是最常用的开源 ODBC API 实现之一。

需要安装的 `unixODBC` 是 `unixODBC` 和 `unixODBC-devel`，然后根据不同的数据库选择驱动。具体的列表可以在 <http://www.unixodbc.org/drivers.html> 上找到，或者也可以用 `yum` 安装：

```
shell> yum install mysql-connector-odbc.
```

配置 `unixODBC` 需要修改 `odbcinst.ini` 和 `odbc.ini` 两个文件。如果不知道它们在哪儿，可以令 `shell> odbcinst -j`。其中 `odbcinst.ini` 中是已经安装的 ODBC 驱动。比如如下的内容：

```
[mysql]
Description = ODBC for MySQL
Driver      = /usr/lib/libmyodbc5.so
```

其中 `[mysql]` 表示的是驱动的名称，`Description` 是对于驱动的说明，`Driver` 指的是驱动的链接库。

而 `odbc.ini` 是用来定义数据源的，比如下面的用法。

```
[test]
Description = MySQL test database
Driver      = mysql
Server      = 127.0.0.1
User        = root
Password    =
Port        = 3306
Database    = zabbix
```

上面的代码很简单，`Description` 又叫做 `DSN` (Data Source Name)，此外就是一些常规的 MySQL 连接配置了。

`unixODBC` 还提供了检查 `odbc.ini` 是否设置正确的工具——`isql`，检查前面写的 `test` 的做法如下。

```
shell> isql test
+-----+
| Connected!                                |
|                                           |
| sql-statement                            |
| help [tablename]                         |
| quit                                     |
|                                           |
+-----+
SQL>
```

这样 `unixODBC` 就都配置好了，`Zabbix` 也需要支持它。在编译的时候加上 `--with-unixodbc` 就行了。

来到前端配置，`Type` 中选择 `Database monitor`，`Key` 的结构是 `db.odbc.select[unique_description,data_source_name]`，其中 `unique_description` 是一个唯一的标识。`data_source_name` 是之前在 `odbc.ini` 中定义的 `DSN`。`SQL query` 是我们跑的 `sql`，`Type of information` 是返回值的类型。

所有超过期限的数据都会被 Housekeeper 删除，笔者建议大家尽可能少地保存 History，Trends 则可以保存比较长的时间。具体时长可以参考数据库的大小来调整。大家可能有疑问了，把 History 的数据都删了，那么怎么看过去的数据呢？图难道是断的？在这方面，Zabbix 会使用 Trends 来填补。如果在 Graph 中把时间段拉长，Zabbix 就会从 Trends 中获取数据。

如果把 History 的保存时间设置为 0，意为 Zabbix 不会保存历史数据，对于监控数据，只会根据数据去检查 Trigger。数据库中也不会存储相应数据，只有表示 Item 最近一次取值的 lastvalue 会记录。

Trends 是 Zabbix 聚合 History 数据的一种机制，和 RRD 的思想类似，它会把老数据聚合计算后存储在数据库——每一小时的最小值、最大值和平均值。和 History 一样，也可以在同样的三个地方设置。一般来说，Trends 保存时间比 History 要长很多。当把保存 Trends 的时间设为 0 的时候，Zabbix 将不会计算 Trends，当然数据库里就不会有这个数据。

RRD (Round Robin Database) 是存储数据的一种特别的方式，使用了环装的结构。定义一个周期，当数据达到这个周期后，就会将老的数据归档压缩（比如采集的时候是 1 分钟一个点，可以压缩成 1 小时一个点），再把新数据写到老数据的位置。这样做是因为针对离现在时间较久的数据，需要的时间粒度较粗。压缩归档可以减少空间消耗，提高查找性能。

5.5 使用 Application 对 Item 分组

Applications 就是 Item 的一个组，类似于 Host group。MySQL Server 可以把所有和 MySQL 相关的 Item 都包含进去。Applications 也是 Web scenarios 的分组，如果使用了 Applications 分组，那么我们在“Monitoring”→“Latest data”中就可以看到 Items 和 Web scenarios 已经被分到了相应的 Applications 中了。

Applications 的配置非常简单，从“Configuration”→“Hosts”或者“Templates”进入，在需要添加 Applications 的 Host 边上单击 Create application，然后输入名字就可以了。

更简单的办法是我们在创建 Item 的时候直接建立，就在 New Application 中输入名字就行了。在 Web scenarios 中，也是一样的方法，在其设置界面中选择即可。

5.6 Item Key详解

Item key 可以有参数，也可以没有参数。比如 Foo 也是一个符合 Zabbix 规范的 key。图 5-7 是 Zabbix 官网中对于 Zabbix Item key 的解析图。在图下方的这根线，跳过了 parameters 的解析，是不带参数的情况，它的上方是 Item key 带有参数的情况，即 key 的名字后面加上方括号，方括号里面是参数，比如在本书开始部分我们使用的 md5sum[FILENAME]。

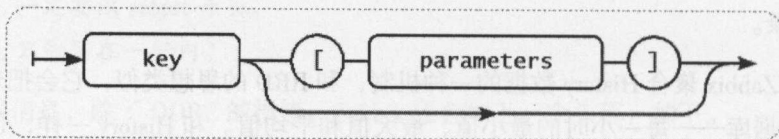


图5-7

Item key 可以是数字、大小写字母、下画线、横线 and 点，即 “0-9a-zA-Z_.”。

Item key 的参数写在方括号中，以逗号分隔。参数可以有引号，也可以没有引号，还可以是一个数组。

对于在每个参数外都加一个引号的参数，如果参数中还需要使用引号，必须在之前加上反斜杠，防止解析出错。如图 5-8 所示。

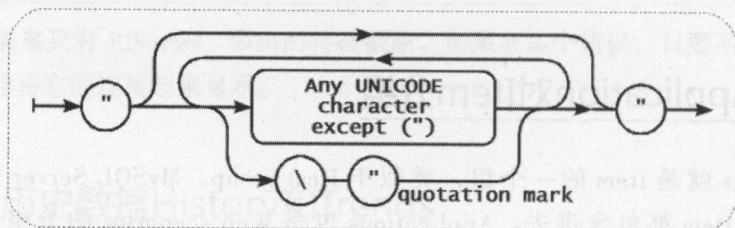


图5-8

对于参数外没有引号的，在参数中不能出现逗号 “,” 和右方括号 “]”，这也是为了防止解析出错。如图 5-9 所示。

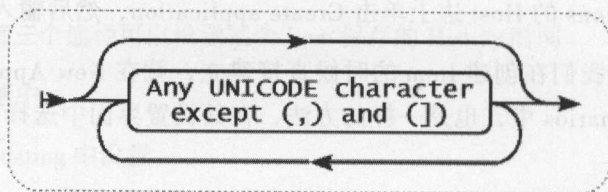


图5-9

5.7 Template模板

模板是一些配置的集合，它可以应用在多个 Host 上，便于配置管理。常用的场景是：有一些 Item 和 Trigger 要应用于一组 Host，但每个 Host 都去添加很多 Item 工作量非常大。我们可以先将这些 Item 和 Trigger 都添加到 Template 上，再用这个 Template 关联到 Host 即可。

在 Template 上可以有以下属性。

- Items
- Triggers
- Graphs
- Applications
- Screens
- Low-level discovery rules
- Web scenarios

Template 是直接关联到 Host 的，不能通过关联 Host group 来关联这个 Host group 下的所有 Host。在 Zabbix 的应用中，一般会有一种服务或者一种类型的服务器制作成一个 Template，比如 MySQL 的 Template、Linux 的 Template 等。

使用 Template 的另一个好处是：当要针对一类 Host 更改配置（比如 Item 的名称）时，可以直接在 Template 上更改，这样只需要修改一个地方即可完成操作。

5.7.1 新建和配置一个Template

在“Configuration”→“Template”中单击“Create template”，创建一个 Template，进入模板标签页，它可以设置的属性有以下几种

- Template name：唯一的模板名称。
- Visible name：设置之后会在 list、map 等显示。
- Groups：Template 属于的 Host group 或者 Template group。
- New Group：新建一个 Group，并且这个 Template 属于它。
- Hosts/Templates：关联到这个模板的 Hosts 和 Templates。

Linked templates 标签页是用来建立模板和模板之间的继承关系的，在这个标签页，可以

选择继承于某一个模板。比如正在建立模板 A，它继承于模板 B，那么 B 的所有设置（包括 Item、Trigger 和 Graph）都会在 A 上产生。只要在搜索框中输入需要的模板，再单击“Add”即可。要取消继承关系，可以单击“Unlink and clear”，这两者的区别在之前说明过，这里不做赘述，忘记的同学可以翻到前面 Host 的地方。

在 Macros 标签页，可以定义一些模板级的 Macro，类似于在 Host 中定义 Host 级的 Macro。

在最下面有几个按钮，分别是“Save”、“Clone”、“Full Clone”、“Delete”、“Delete and clear”、“Cancel”。这里主要讲一下“Delete”和“Delete and clear”。

◎ Delete：删除 Template 后，和 Template 关联的 Host 保留 Template 上的 Item 等属性。

◎ Delete and clear：删除 Template 后，和 Template 关联的 Host 也会删除相应的 Item 等属性。

必须先在 Template 上建立 Item，然后才能建立相应的 Trigger 和 Graph。Trigger 和 Graph 没办法脱离 Item 直接和 Template 建立关联。

在 Template 上添加 Item、Graph 等都和在 Host 上添加一样，下面以添加 Item 为例。从菜单栏中进入“Configuration”→“Template”，单击需要添加 Item 的 Template，再单击右上角的 Create Item 即可，如图 5-10 所示。

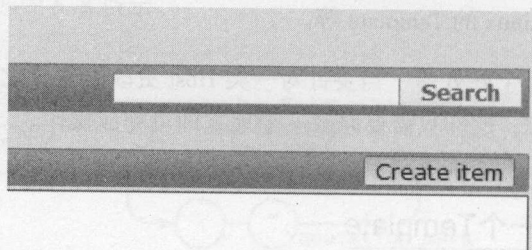


图5-10

除了直接在 Template 上创建外，还可以将现有的 Item 或者其他属性直接复制到 Template，操作也很简单：首先找到并选中需要的 Item，然后选择页面底部的“Copy selected to...”，单击旁边的“Go”按钮（Go 右边会显示目前选中的数目），在“Target type”中选择“Hosts”，“Group”中选择 Template 所在的 Group，如图 5-11 所示。这里可能有读者有疑问，明明是 Template，为什么“Target type”却是“Hosts”呢？因为，Template 可以理解为一个 Host，在数据库中，它们是存在于同一张表——Hosts 中的。

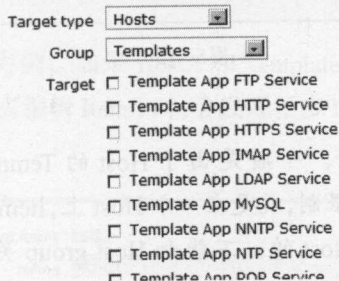
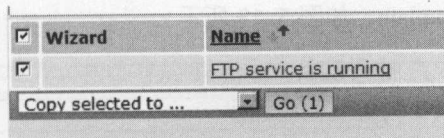


图5-11

对于 Template 的其他操作，都可以在“Configuration”→“Template”里进行，单击需要修改的 Template 后，在标签页中切换操作。

5.7.2 建立/取消Host和Template的关联

前面介绍了怎样去配置一个 Template，但它自己是没法工作的，要将它和 Host 放在一起。Host 和 Template 建立关联有两种方法：一种是一个 Host 关联到多个 Template，另一种是一个 Template 关联到多个 Host。

首先看看如何把一个 Host 关联到多个 Template。

找到需要的 Host，单击进入配置后，选择 Templates 标签页，然后在“Link new templates”中输入需要的 Template，最后单击“Add”按钮就行了。如图 5-12 所示。

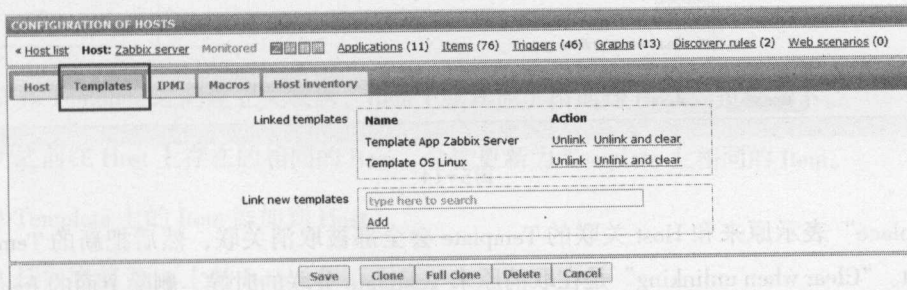


图5-12

还可以一次添加多个 Template。如图 5-13 所示。

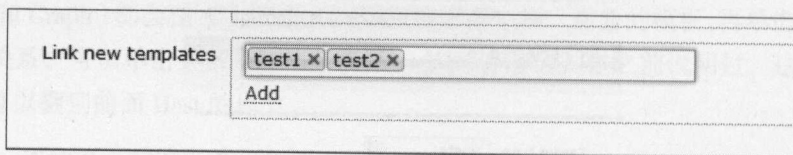


图5-13

注意：

(1) 如果在 Save 时，有报错，一般是由于 Host 的 Template 上，有 Item 的 Key 相同。Zabbix 对于 Host 上的 Item 有一个限制，就是在一个 Host 上，Item Key 是独一无二的，不能重复。

(2) Template 是直接关联到 Host 的，不能和 Host group 关联。前面提到过，并不是把 Template 放在某个 Host group 中，这个 Host group 中的所有机器就自动和它关联。Host group 只是一个非常松散的概念，它的唯一用处就是对 Host 有一个逻辑上的分组。

下面看看如何将一个 Template 关联到多个 Host。

选中 Template，在右侧的“OtherGroup”中选择 Host group，它下面的框会显示选中 Host group 中的 Host，选中（可以使用【Ctrl】键和【Shift】键进行多选）需要的 Host，单击“<<”就行了。

在 Zabbix 中，这种两个框的操作非常多，左边的表示选中的和将会添加的（或类似意思），右边则是供选择的，而中间的“<<”和“>>”就是将选中的元素在两个框中进行移动。

还可以选中多个 Host 后，选择页面下方的“Mass update”，然后在标签页中选择需要的 Template。如图 5-14 所示。

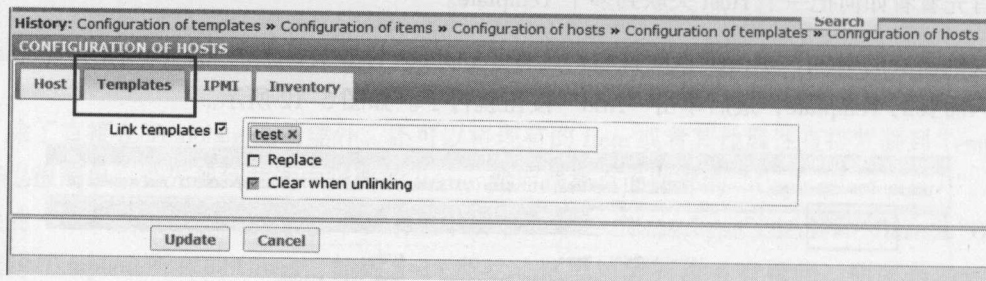


图5-14

“Replace”表示原来和 Host 关联的 Template 会全部被取消关联，然后把新的 Template 关联到 Host。“Clear when unlinking”是在取消原来 Template 关联的时候，删除上面的 Item。

取消 Host 和 Template 的关联, 和添加关联的操作是类似的, 比如要取消 Host 和 Template 的关联, 就在前面添加关联的地方, 在已有的 Template 后面单击 “Unlink” 或者 “Unlink and clear”。

5.7.3 修改 Template

以修改 Template 的 Item 为例, 如果 HostA 和 TemplateA 关联, Template 会有一个 Item 叫做 ItemA。当直接在 Host 层面去编辑 ItemA 时, 会发现很多 Item 的属性是无法编辑的, 如图 5-15 所示。

The screenshot shows the configuration form for a Zabbix item. The fields are as follows:

- Parent items: test1
- Name: echo1
- Type: Zabbix agent
- Key: systemm.run["echo 1"]
- Host interface: 127.0.0.1 : 10050
- Type of information: Numeric (unsigned)
- Data type: Decimal
- Units: (empty)
- Use custom multiplier: ☒ 1
- Update interval (in sec): 30
- Flexible intervals: A table with columns Interval, Period, and Action. The content is "No flexible intervals defined."

图5-15

因为 ItemA 是由于 TemplateA 才和 HostA 产生关联的, 不能直接在 Host 上修改, 只能在 Template 层面上修改, 这个修改是针对所有关联这个 Template 的 Host 上面的 Item 的。我们能在 Host 上做的, 就是 Enable 和 Disable 某个 Item。

5.7.4 Template和Host

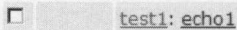
Host 和 Template 之间建立关联后, Item (或其他) 添加到 Host 的过程如下。

- (1) 之前在 Host 上存在的相同的 Item, 会被更新为 Template 上相同的 Item。
- (2) Template 上的 Item 添加到 Host。
- (3) 任何之前 Host 上的 Item, 保持不变。

那么，Zabbix 是如何判断 Host 上原有的 Item（或其他）是否和 Template 上的是一致的呢？对于不同的项目，有不同的判断方法。

- （1）Item：根据 Item 的 key。
- （2）Trigger：根据 Trigger 的名字和 Expression。
- （3）Custom graph：根据 Graph 的名字，和它包含的 Item。
- （4）Application：根据 Application 的名字。

在查看 Host 的时候，所有 Template 上的 Item，都会有一个前缀，可以据此了解到 Item 是因为 Host 关联了哪个 Template 而添加到 Host 的，如图 5-16 所示。



☐ test1: echo1

图5-16

Zabbix 自带了很多 Template，其中很多都能直接使用或者参考，但是需要注意的是：Zabbix 自带的监控模板，可能会因 Item 的数目过多或 Interval 太短而影响性能，需要修改一下。

5.7.5 Template之间的父子关系

在 Zabbix 中，也支持 Template 和 Template 的关联关系，称为 Template 之间的父子关系。

这个比较适合这样的应用场景，比如有 Template_Linux（系统监控）、Template_Network（网络监控）和 Template_MySQL（应用上的监控），对于某一个机房，它部署的服务器都是需要这三个 Template 的，那么就可以建立一个 TemplateIDC，前面三个 Template 和这个 TemplateIDC 关联，当把 Host 和 TemplateIDC 关联后，就自动把 Template_Linux、Template_Network 和 Template_MySQL 关联到 Host 上。

将 TestTemplateA 作为 TestTemplateB 的父模板，配置的步骤如下。

- （1）打开 TestTemplateA 配置界面。
- （2）选择 Linked templated 标签。
- （3）选择 TestTemplateB，单击“Add”按钮。
- （4）单击“Save”按钮。

这样，只要 Host 和 TestTemplateA 关联，那么就自动把 TestTemplateA 也关联到 Host 上。在 Host 的属性界面，是看不出 TestTemplateA 和 TestTemplateB 的父子关系的，如图 5-17 所示。

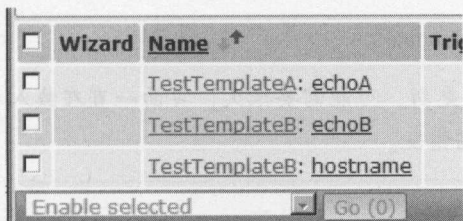


图 5-17

在此基础上能做到三个 Template 的两层父子关系，只需要再编辑 TestTemplateB，使其作为 TestTemplateC 的父模板即可。图 5-18 为 Template 之间的关系和 Host 上的 Item。

<input type="checkbox"/> Templates	Applications	Items	Triggers	Graphs	Screens	Discovery	Web	Linked templates
<input type="checkbox"/> TestTemplateC	Applications (0)	Items (1)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	-
<input type="checkbox"/> TestTemplateB	Applications (0)	Items (3)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	TestTemplateC
<input type="checkbox"/> TestTemplateA	Applications (0)	Items (4)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	TestTemplateB

<input type="checkbox"/> Wizard	Name	Trigger
<input type="checkbox"/>	TestTemplateA: echoA	
<input type="checkbox"/>	TestTemplateB: echoB	
<input type="checkbox"/>	TestTemplateC: echoC	
<input type="checkbox"/>	TestTemplateB: hostname	

图 5-18

5.8 Clone、Full Clone和Mass Update

在 Zabbix 配置中，大多数的配置都有 Clone 和 Full Clone。以 Host 为例，当从一个 Host 复制一个 Host 出来的时候，会将源 Host 的所有参数和模板的连接都保留，包括模板上的 Graph、Trigger 等。Full Clone 除了 Clone 的功能，还会将直接与源 Host 关联的项目复制过来，比如直接挂在 Host 上的 Graph、Trigger 等。

Mass update 是更新大批量 Item 时的利器。具体操作步骤如下。

(1) 在需要更新的 Item 前的选择框上打勾；

(2) 在最下面的下拉列表选择 Mass update，然后单击 Go；

(3) 在弹出的窗口中选择需要更新的项目；

(4) 输入新的值，单击 update。

注意：对于大量数据的更新，可能需要很久，页面一直在载入状态，这时不要取消，否则会造成 Zabbix 数据库中数据错乱。

5.9 Windows监控

一般来说，互联网公司的服务器以 Linux 居多，但也有一些游戏公司用的是 Windows 机器，下面就看下 Zabbix 支持的对于 Windows 机器的 Item。

Zabbix 使用 perf_counter[] 这个 key 来监控 Windows，比如 perf_counter["\\Processor(0)\\Interrupts/sec"] 这种形式。

设置语言不同的 Windows 机器的 key 是不同的（比如这里英语是 Processor(0) Interrupts/secs），所以使用一个模板去适配所有的 Windows 机器很有可能会因为语言的问题而出问题。但是可以在 Windows 注册表中设定这个 key 和一个数字的映射关系。这是和语言无关的，在监控时只要使用这个数字就可以了。具体是在注册表的 HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\WindowsNT\\CurrentVersion\\Perflib\\009 中。比如输入如下的值。

1	
1847	
2	
System	
4	
Memory	
6	
% Processor Time	
10	
File Read Operations/sec	
12	
File Write Operations/sec	

```

14
File Control Operations/sec
16
File Read Bytes/sec
18
File Write Bytes/sec
....

```

对于“\System\% Processor Time”这个 key 来说, 由于映射表的存在, System 对应 2, % Processor Time 对应 6, 所以这个 key 对应的就是 \2\6。

也可以在 Zabbix Agent 的配置文件中定义一些 PerfCounter, 这样在前端使用 Item Key 时更加简单。比如根据之前的映射关系, 可以进行如下定义。

```
PerfCounter=UserPerfCounter1," \Memory\Page Reads/sec",30
```

或者是

```
PerfCounter=UserPerfCounter2," \4\24",30
```

此时, 就能使用 UserPerfCounter1 或者 UserPerfCounter2 来创建相应的 Item 了。记得修改配置文件或重启 Zabbix Agent 使其生效。

Zabbix 自带了两个监控 Windows 的 Template, 不懂 Windows 运维的人, 直接把模板套上就可以了。

最后看看 zabbix_agentd.exe 可以设置的参数, 如下。

```

C:\zabbix-2.2.0\bin\win64>zabbix_get.exe --help
Zabbix get v2.2.0 (revision 40147) (12 November 2013)
usage: zabbix_get.exe [-hV] -s <host name or IP> [-p <port>] [-I <IP address>]
      -k <key>
Options:
-s --host <host name or IP>          Specify host name or IP address of
-p --port <port number>              Specify port number of agent running
                                     on the host. Default is 10050
-I --source-address <IP address>     Specify source IP address
-k --key <key of metric>             Specify key of item to retrieve value

```

```

-h --help                Give this help
-V --version             Display version number
Example: zabbix_get -s 127.0.0.1 -p 10050 -k "system.cpu.load[all,avg1]"

```

使用 Typeperf 监控 Windows

在 Zabbix 自带的 Template OS Windows 中，有很多 Item 的 key 使用的是 perf_counter，比如监控平均磁盘读队列长度的“perf_counter[\234(_Total)\1402]”。Zabbix 90% 的使用场景都是 Linux 的，对于 Windows 基本处于用阶段。一些公司有一小部分 Windows 服务器需要监控，也需要加入 Zabbix。下面向大家介绍一下如何使用 Perf_counter 收集 Windows 的数据。

Perf_counter 的全称是 Performance Counters，MSDN（微软开发者社区，Microsoft Developer Network）对它的解释是“它用来监控操作系统、应用、服务或者驱动程序的运行情况。counter 数据可以帮助我们检查系统或者应用的瓶颈。这些数据可以帮助用户以图形的形式来掌握系统的运行状况”，简单地说，Windows 提供了一套接口，使得我们可以获取 Windows 的系统性能数据，这套接口就叫做 Performance Counters。

从“开始”→“管理工具”→“性能监视器”进入图形化界面，如图 5-19 所示，在左侧选择性能监视器，就能看到默认显示的“% Processor Time”了。

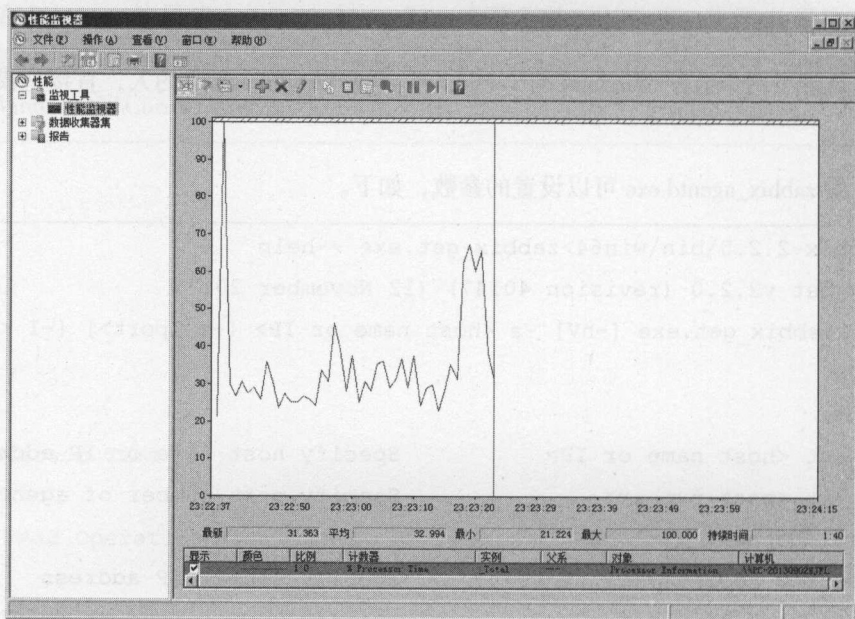


图5-19

我们已经了解 perf_counter 的作用了,现在需要知道 perf_counter 后面加什么参数,即“perf_counter[234(_Total)\1402]”中的“\234(_Total)\1402”。有两种方法:一种是使用“性能监视器”,另一种是使用命令行。

使用“性能监视器”来获取想要监控数据的 key 时,首先单击如图 5-20 所示的加号。

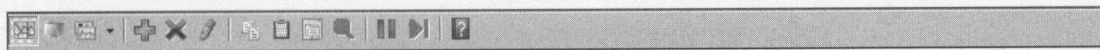


图5-20

在打开的窗口中选择想要添加的监控项,如图 5-21 所示。

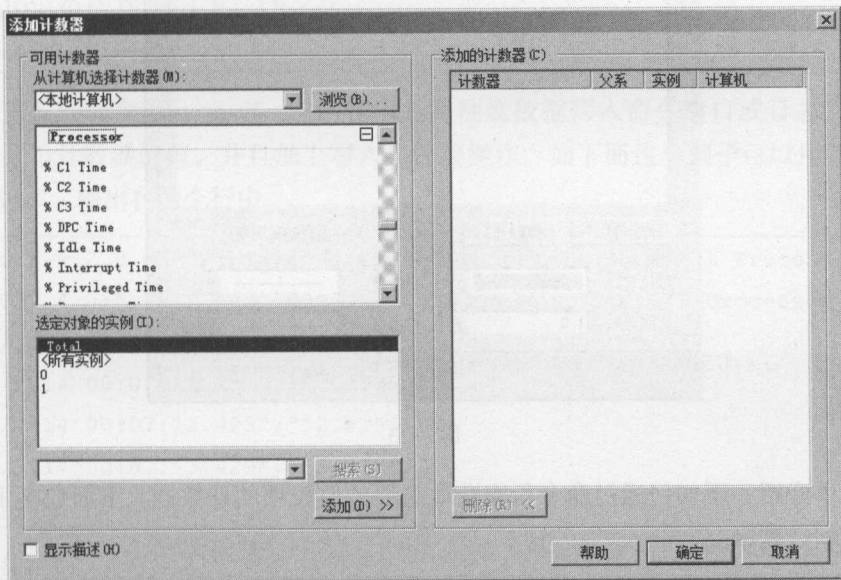


图5-21

在左侧可以选择监控的大类,单击加号可展开,如图 5-21 中选择的“Processor”大类。在选择了监控项后,下面还有参数可以选择。笔者在图 5-21 中选择的是处理器“Processor”,因为是双核的 CPU,所以可以选择监控哪个核心,即“Total”、“<所有实例>”、“0”和“1”。选择需要的监控项后就单击“添加”按钮,最后单击“确定”按钮,就能看到两条线了,并且在窗口下面能看到如图 5-22 所示的显示。

显示	颜色	比例	计数器	实例	父系	对象	计算机
		1.0	% Processor Time	Total	---	Processor Information	\\PC-20130902WJFL
		1.0	% C2 Time	Total	---	Processor	\\PC-20130902WJFL

图5-22

右键单击任意一个，就可以查看目前显示的两个数据的计数器的值了，如图 5-23 所示。

这就是我们要在 Zabbix 的 perf_counter 后面填写的参数。

看完了图形化界面，肯定会有读者觉得这个是非常麻烦并且反人类的设计，如果有命令行模式就好了。Performance Counters 还真支持 Windows 命令行。

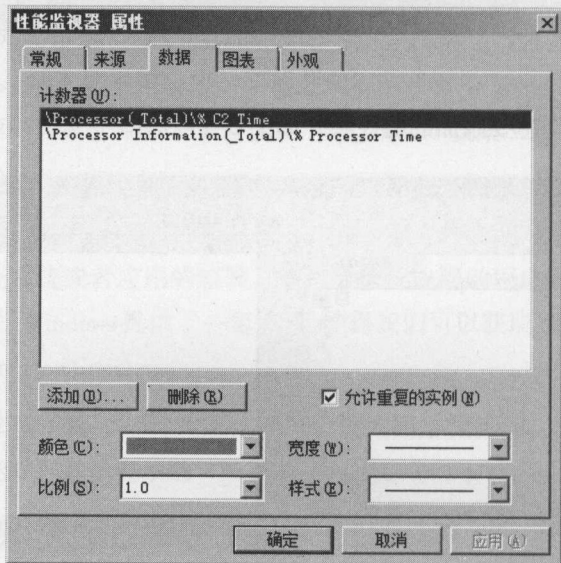


图5-23

下面简单介绍一下如何通过命令行来获取想要的监控项的计数器。下面包括了所有带有“disk read”的计数器。

```
C:\zabbix-2.2.0\bin\win64>typeperf -qx | find /I "disk read"
\\LogicalDisk(C:)\% Disk Read Time
\\LogicalDisk(D:)\% Disk Read Time
\\LogicalDisk(_Total)\% Disk Read Time
\\LogicalDisk(C:)\Avg. Disk Read Queue Length
\\LogicalDisk(D:)\Avg. Disk Read Queue Length
\\LogicalDisk(_Total)\Avg. Disk Read Queue Length
\\LogicalDisk(C:)\Disk Reads/sec
\\LogicalDisk(D:)\Disk Reads/sec
\\LogicalDisk(_Total)\Disk Reads/sec
```

```

\LogicalDisk(C:)\Disk Read Bytes/sec
\LogicalDisk(D:)\Disk Read Bytes/sec
\LogicalDisk(_Total)\Disk Read Bytes/sec
\PhysicalDisk(0 C: D:)\% Disk Read Time
\PhysicalDisk(_Total)\% Disk Read Time
\PhysicalDisk(0 C: D:)\Avg. Disk Read Queue Length
\PhysicalDisk(_Total)\Avg. Disk Read Queue Length
\PhysicalDisk(0 C: D:)\Disk Reads/sec
\PhysicalDisk(_Total)\Disk Reads/sec
\PhysicalDisk(0 C: D:)\Disk Read Bytes/sec
\PhysicalDisk(_Total)\Disk Read Bytes/sec

```

这里最重要的命令是 `typeperf`，它的作用是将性能数据写入命令窗口或日志文件。可以用它来监控某个计数器的值，并且加上写入文件等操作。如下面这个例子可以用于它来监控“Processor Time”，输出在命令行中。

```

C:\zabbix-2.2.0\bin\win64>typeperf "\Processor(_Total)\% Processor Time"
"(PDH-CSV 4.0)","\\PC-20130902WJFL\Processor(_Total)\% Processor Time"
"03/24/2014 00:07:10.419","19.824433"
"03/24/2014 00:07:11.421","10.572836"
"03/24/2014 00:07:12.425","12.215615"
"03/24/2014 00:07:13.429","11.526874"
"03/24/2014 00:07:14.433","15.932421"

```

命令成功结束。

这里使用的是它的 `qx` 参数，来列出所有支持的计数器。再加上 Windows 中的 `find` 工具，就可以找到我们需要的了。`find` 工具和 Linux 中的 `grep` 类似，可以用 `help` 命令来查看用法，如下。

```

C:\zabbix-2.2.0\bin\win64>help find

```

用法如下。

```

FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] "string" [[drive:][path]filename[
...]]

```

其中各部分含义如下。

- ◎ /V : 显示所有未包含指定字符串的行。
- ◎ /C : 仅显示包含字符串的行数。
- ◎ /N : 显示行号。
- ◎ /I : 搜索字符串时忽略大小写。
- ◎ /OFF[LINE] : 不要跳过具有脱机属性集的文件。
- ◎ "string" : 指定要搜索的文本字符串。
- ◎ [drive:][path]filename : 指定要搜索的文件。

如果没有指定路径，FIND 将搜索在提示符处键入的文本或者由另一命令产生的文本。

注意：Windows 中命令的参数是加 “/” 符号的，而不是 Linux 中的 “-”。

5.10 VMware监控

从 Zabbix 2.0 开始，Zabbix 支持对 VMware 的监控，Zabbix 可以使用 low-level discovery 自动寻找 VMware hypervisors 和虚拟机，并且能根据事先定义的，为这些虚拟机建立 Host，添加监控。Zabbix 默认提供了一些模板，可以直接用来监控 VMware、vCenter（中心节点，控制虚拟机的控制访问等）和 vSphere（VMware 虚拟机系统）。

Zabbix 是通过叫做 “vmware collector” 的进程来监控 VMware 的，它和 VMware web 服务器通过 SOAP 协议进行通信，并且将获取的信息存到 Zabbix 内存中，然后就可以从虚拟机中抽取数据了。

从单个 VMware 服务抓取数据时只会使用一个 vmware collector 进程。

如果要 Zabbix 支持对虚拟机的监控，需要在编译 Zabbix Server 时，加上 libxml 和 libcurl 的依赖。在 zabbix_server.conf 中，和虚拟机监控有关的有如下三个配置。

- ◎ StartVMwareCollectors : vmware collector 的进程数。
- ◎ VMwareCacheSize : 保存 VMware 数据的内存空间。这些内存空间直到 vmware collector 启动时才会分配。
- ◎ VMwareFrequency : 对同一个 VMware 服务抓取数据的时间间隔。这个参数最少要和 VMware 监控的 Item 的时间间隔相等。

Zabbix 可以使用 low-level discovery 来将虚拟机加入监控，只需要将 Discovery rule 中的 Key 设置为 vmware.hv.discovery[{\$URL}] 即可。

在 lld 中，可以使用很多宏来定义原型，如图 5-24 所示，可以根据 Hypervisor 的宏来定义 Host prototype。

图 5-24

表 5-1 是在 Discovery 中支持的 key 和宏。

表 5-1

Key	Key 说明	宏	宏说明
vmware.cluster.discovery	对于 VMware 集群的 discovery	{#CLUSTER.ID}	集群的 ID
		{#CLUSTER.NAME}	集群的名字
vmware.hv.discovery	对于 Hypervisor 的 discovery	{#HV.UUID}	唯一的 Hypervisor 的 ID
		{#HV.ID}	Hypervisor 的 ID (由 HostSystem 管理)
		{#HV.NAME}	Hypervisor 的名字
		{#CLUSTER.NAME}	集群名字, 可能为空
vmware.hv.datastore.discovery	对于 Hypervisor datastore 的 discovery。 多个 Hypervisor 可能使用同一个 datastore	{#DATASTORE}	Datastore 的名字

续 表

Key	Key 说明	宏	宏说明
vmware.vm.discovery	VMware 虚拟机的 discovery	{#VM.UUID}	虚拟机的 UUID
		{#VM.ID}	虚拟机的 ID
		{#VM.NAME}	虚拟机的名字
		{#HV.NAME}	Hypervisor 的名字
		{#CLUSTER.NAME}	集群的名字, 可能为空
vmware.vm.net.if.discovery	VMware 虚拟机的网卡 discovery	{#IFNAME}	网卡的名字
vmware.vm.vfs.dev.discovery	VMware 虚拟机磁盘设备 discovery	{#DISKNAME}	磁盘设备的名字
vmware.vm.fs.discovery	VMware 虚拟机文件系统 discovery	{#FSNAME}	文件系统的名字

5.11 Zabbix监控性能

Zabbix 的性能，可以在“Administration”→“Queue”查看，这里所说的 Queue，指的是等待被刷新的 Item，这只是一个逻辑上的表述，并不是 IPC queue 或其他 Zabbix 的 Queue 机制。

Queue 相关的数据是反应 Zabbix Server 性能情况的重要指标，可以从“Administration”→“Queue Overview”进入查看 Queue 数据。如图 5-25 所示。

Items	5 seconds	10 seconds	30 seconds	1 minute	5 minutes	More than 10 minutes
Zabbix agent	0	1	0	0	1	0
Zabbix agent (active)	0	0	0	0	0	0
Simple check	0	0	0	0	0	0
SNMPv1 agent	0	0	0	0	0	0
SNMPv2 agent	0	0	0	0	0	0
SNMPv3 agent	0	0	0	0	0	0
Zabbix internal	0	0	0	0	0	0
Zabbix aggregate	0	0	0	0	0	0
External check	0	0	0	0	0	0
Database monitor	0	0	0	0	0	0
IPMI agent	0	0	0	0	0	0
SSH agent	0	0	0	0	0	0
TELNET agent	0	0	0	0	0	0
JMX agent	0	0	0	0	0	0
Calculated	0	0	0	0	0	0

图5-25

实际界面上绿色背景的指标表示这个数据是在正常范围之内,从图 5-25 中看到有一个 Item 延迟了 10 秒,还有一个 Item 延迟了 5 分钟,从 Details 进入后,可以找到到底是哪些 Items 延迟了。

有少数几个延迟是正常的,不需要去在意,只有当大量 Item 延迟时,才应该去看看 Zabbix Server 或者 Zabbix Agent 是否需要检查。需要注意的是,Child Node (Zabbix 分布式架构中的子节点)的 Queue 信息是不准确的,因为 Master 节点获取 Child 节点数据是有一定时间延迟的。Child 节点的 Queue 信息还和 Child Node 性能有关,Child 和 Master 的网络连通性也和它有关,此外,两个节点的本地时间的不同也可能会有影响。

对于监控 Queue 信息,有一个特殊的 internal item 可以使用: `zabbix[queue,<from>,<to>]`,它会返回 `<from>` 和 `<to>` 之间有多少数据延迟了。

第 6 章

报警配置

6.1 Triggers

作为一个监控系统，光是收集监控数据是不够的，还需要针对数据进行报警，Triggers 起的就是这个作用。简单地说，它告诉 Zabbix，一个 Trigger 的状态是好还是不好，是正常还是不正常。Trigger 的逻辑判断条件可以是非常多的条件的组合，比如“CPU 负载大于 10，并且磁盘空余空间小于 10%”。一个 Trigger 可能对应了多个 Item，那么在每一个 Item 的数据到达 Zabbix 的时候，Zabbix 都要去重新判断这个 Trigger 的状态。

在 Zabbix 中，Trigger 的状态一共有两个：一个是“OK”，一个是“Problem”，也就是很早之前的“False”和“True”。在 1.8.X 版本的时代，Trigger 还有个状态叫做 Unknown，Trigger 状态从“OK”变到“Unknown”是不会报警的，这是一个好大的 bug。当然现在 2.X 时代已经修复了。有兴趣的可以看一下：<https://support.zabbix.com/browse/ZBXNEXT-341>。

6.1.1 配置 Triggers

Trigger 一定要挂靠在一个 Host 上，从“Configuration”→“Hosts”找到一台 Host，然后在它上面添加一个 Trigger 即可。

标签“Trigger”就是关于 Trigger 的设置了，它包括以下属性。

◎ Name：Trigger 的名字，可以使用 {HOST.HOST}、{HOST.NAME}、{HOST.CONN}、{HOST.DNS}、{HOST.IP}、{ITEM.VALUE}、{ITEM.LASTVALUE}、{\$MACRO}。同时还支持 \$1，\$2

等占位符，它表示的是在 Expression 里的常量。当 Name 设置为 “Processor load above \$1 on {HOST.NAME}”，而 Expression 是 “New host:system.cpu.load[percpu,avg1].last(0)>5” 时，Name 就会变成 “Processor load above 5 on New host”。读者可能有疑问，为什么 \$1 不是 “last(0)” 里的 “0”，而是 5 呢？大家可以在下一节对 expression 的介绍中找到答案。

- ◉ Expression : Trigger 的判断逻辑，是 Trigger 的核心。
- ◉ Multiple PROBLEM events generation : 如果勾选此项，那么每一次 Trigger 检查为 Problem 时都会生成一个 event。如果不勾选，只有当 Trigger 状态变化时才会生成 event。
- ◉ Description: 关于这个 Trigger 的一些描述。可以在里面写对于这个 Trigger 的处理方法等。从 Zabbix2.2 开始，Description 中可以使用和 Name 一样的宏。
- ◉ URL : 如果这里不是空的话，在 Monitoring-Triggers 中，单击 Trigger 就可以访问这个 URL，这里可以使用 {TRIGGER.ID} 宏。
- ◉ Severity : 重要性，可以设置不同的等级。
- ◉ Enabled : 如果不勾选，这个 Trigger 是不生效的。

Dependencies 标签页会在下面几个小节介绍。

6.1.2 Trigger expression

Expression 是 Trigger 的核心，它定义了 Trigger 的判断逻辑。它非常灵活，可以组合出各种各样复杂的逻辑，基本形式如下。

{<server>:<key>.<function>(<parameter>)}<operator><constant>

以前面的 “New host:system.cpu.load[percpu,avg1].last(0)>5”，把它和上面的公式对照起来，如表 6-1 所示。

表 6-1

<server>	New host	Host 的 hostname
<key>	system.cpu.load[percpu,avg1]	Item
<function>	last	最近的数据
<parameter>	0	last 的参数，表示最近的一次数据
<operator>	>	大于
<constant>	5	5

对于大多数 function 来说，它接受的数字的单位是“秒”，如果数字前有“#”，则表示的是次数。

比如 `sum(600)` 表示的是过去 600 秒的数据的和, 而 `sum(#5)` 表示的是最近 5 次的数据的和。注意, `last` 这个 function 的 “#” 的含义有些特殊, 它后面的数字表示的是 “第几个” 数据, 比如有数据 3,7,2,6,5, 那么 `last(#2)` 表示的是 7, `last(#5)` 表示的是 5。

对于不需要 parameter 的 function, 也要加上, 不能省略。

`avg`、`count`、`last`、`min` 和 `max` 这 5 个 function 除了支持一个 parameter, 还支持一个参数——`time_shift`, 它表示这个函数计算的数据需要往前推一段时间。比如 `avg(1h,1d)` 表示 “1 天前的前 1 个小时的数据的平均值”。如果现在是 12 月 20 号的 20 点, 那么 `avg(1h,1d)` 计算的就是 12 月 19 号的 19 点到 20 点的数据的平均值。

Operators 很容易理解, 是符号类型, 表 6-2 是 Expression 中支持的 operator (优先级依次递减)。

表 6-2

优先级	符 号	说 明
1	/	除
2	*	乘
3	-	减
4	+	加
5	<	小于, 这里不是严格意义的小于, $A < B$ 只要 $A \leq B - 0.000001$
6	>	大于, $A > B$ 只要 $A \geq B + 0.000001$
7	#	不等于, $A \neq B$ 的充分必要条件是 $(A \leq B - 0.000001) \vee (A \geq B + 0.000001)$
8	=	等于, $A = B$ 的充分必要条件是 $(A > B - 0.000001) \wedge (A < B + 0.000001)$
9	&	逻辑与, 即 AND
10		逻辑或, 即 OR

下面看几个具体的例子。

(1) `{HOSTNAME1:system.cpu.load[all,avg1].last(0)}>5` : HOSTNAME1 这台机器的 CPU 负载 (`system.cpu.load[all,avg1]`) 最近一次的值大于 5。

(2) `{HOSTNAME1:system.cpu.load[all,avg1].last(0)}>5|{HOSTNAME1:system.cpu.load[all,avg1].min(10m)}>2` : HOSTNAME1 的 CPU 负载最近一次获取的值大于 5 或者最近 10 分钟的最小值大于 2。

(3) `{HOSTNAME1:vfs.file.cksum[/etc/passwd].diff(0)}>0` : HOSTNAME1 的 `/etc/passwd` 文件发生了变化 (`vfs.file.cksum` 获取文件 md5sum 的命令), 同样可以监控 Linux 比较重要的文件是否

被更改。

(4) {HOSTNAME1:net.if.in[eth0,bytes].min(5m)}>100K : HOSTNAME1 的 eth0 网卡的入口流量在过去 5 分钟内的最小值大于 100KB。

(5) {HOSTNAME1:net.tcp.service[smtp].last(0)}=0 & {HOSTNAME2:net.tcp.service[smtp].last(0)}=0 : HOSTNAME1 和 HOSTNAME2 的 SMTP 服务都出了问题。

(6) {HOSTNAME1:agent.version.str("beta8")}=1 : HOSTNAME1 的 agent 版本是 “beta8”。

有一些 Trigger 的 OK、Problem 状态是跟 Trigger 的值有关的, 比如, 想定义一个监控 IDC 机房温度的 Trigger, 当温度超过 20℃ 的时候 Trigger 要变成 Problem, 直到温度低于 15℃ 时, 才会变回 OK。如果按照之前的做法, {HOSTNAME1:temperature.last(0)}>20, 那么当温度达到 21℃ 时, Trigger 确实会变为 Problem, 但当温度回到 19℃ 并且保持的时候, Trigger 就会变成 OK。但是这样 IDC 机房是不是恢复正常了呢? 当然不是。我们需要的是 {TRIGGER.VALUE} 这个宏, 它在 Trigger 是 OK 时值为 0, Problem 时值为 1, 具体的 Expression 如下。

```
{(TRIGGER.VALUE)=0 & {server:temp.last(0)}>20} |
{(TRIGGER.VALUE)=1 & {server:temp.last(0)}>15}
```

这个 Expression 表示 Trigger 变成 Problem 状态的条件是: 当 Trigger 是 OK 时温度大于 20℃ 或者当 Trigger 是 Problem 时温度大于 15℃。

更进一步, 在磁盘空间连续 5 分钟小于 10GB 的情况下, Trigger 变为 Problem, 当磁盘空间连续超过 10 分钟大于 40GB 的情况下, Trigger 变成 OK。expression 如下。

```
{(TRIGGER.VALUE)=0 & {server:vfs.fs.size[/,free].max(5m)}<10G} |
{(TRIGGER.VALUE)=1 & {server:vfs.fs.size[/,free].min(10m)}<40G}
```

6.1.3 Function 详解

上一节我们学习了 Triggers 里很关键的一个 Function, 它会根据输入返回一些值。本节会把 Zabbix 支持的所有 Function 都整理一遍, 帮助读者使用。

使用 “[float|int|str|text|log].abschange” 这种格式来定义一个 function, 说明 abschange 可以在 float、int、str、text 和 log 类型的 Item 上使用, 不需要任何参数。下面具体看看 Zabbix 支持的 Function。

© [float|int|str|text|log].abschange(): 表示前两次的值的绝对值是否相同。0 表示相同, 1 表示不同。

- ◎ `[float|int].avg(sec|num, time_shift)`: 一段时间内, 或者几次取值的平均值。参数可以是秒数, 可以是次数, #5 表示最近 5 次的的数据。第二个参数可选, 表示的是 `time_shift`, 详情可以看上一节的说明。
- ◎ `int.band(sec|num, mask, time_shift)`: 和 `mask` 中定义的数字做二进制与操作。
- ◎ `[float|int|str|text|log].change()`: 最近一次和上上次获取的值是否不同, 0 表示相同; 1 表示不同。
- ◎ `[float|int|str|text|log].count(sec|num, pattern, operator, time_shift)`: 在一段时间内或者几次收集到的数据中符合条件的个数。

其中对于 `pattern`, “integer items” 表示数据要完全吻合, “float items” 表示只需要误差在 0.000001 就认为是相等。

`operator` 支持的有:

- `eq` - equal: 相等。
- `ne` - not equal: 不相等。
- `gt` - great than: 大于。
- `ge` - greater or equal: 大于等于。
- `lt` - less than: 小于。
- `le` - less or equal: 小于等于。
- `like` - 如果监控值包含 `pattern`: 则计数 +1, 主要用于文本类的 Item。
- `band` - bitwise AND。从 Zabbix 2.2.0 开始支持。如果使用了 `band`, 那么 `pattern` 可以使用 “`number_to_compare_with/mask`”。如果 bitwise AND 后的值和 `number_to_compare_with` 相等, 那么这次会使得计数 +1。如果 `number_to_compare_with` 和 `mask` 相等, 只需要写一个 `mask` 就行了。

对于 integer items, operators 支持的有 `eq` (默认)、`ne`、`gt`、`ge`、`lt`、`le`、`band`; 对于 float items, operators 支持的有 `eq` (默认)、`ne`、`gt`、`ge`、`lt`、`le`; 对于 string、text 和 log items, operators 支持的有 `like` (默认)、`eq` 和 `ne`。

下面看几个例子。

`count(600)`: 过去 600 秒数据的个数。

`count(600, 12)`: 过去 600 秒内, 监控值为 12 的个数。

`count(600,12," gt")` : 过去 600 秒内, 监控值大于 12 的个数。

`count(#10,12," gt")` : 过去 10 次数据中大于 12 的个数。

`count(600,12," gt" ,86400)` : 在一天前 (86400 秒) 的这个时间点的过去 600 秒内, 大于 12 的监控值的个数。

`count(600,6/7," band")` : 在过去 600 秒内, 监控值二进制后的最后三位是 110 (十进制 6) 的监控值的个数。

`count(600,,,86400)` : 在一天前 (86400 秒) 的这个时间点的过去 600 秒内的监控值的个数。

- `any.data()` : 返回当前的日期, 格式为 YYYYMMDD, 如 “20131025”。
- `any.dayofmonth()` : 返回日期, 返回值的范围是 1 ~ 31。
- `any.dayofweek()` : 返回星期几, 返回值的范围是 1 ~ 7。
- `[floatlint].delta(sec|#num,time_shift)` : 在 sec 秒内或者 #num 次内获取数据最大值和最小值的差
- `[floatlint|str|text|log].diff()` : 判断上一次和上上次获取的数据是不是相同, 1 表示不同, 0 表示相同。
- `[floatlint].fuzzytime(sec)` : 返回返回值和 Zabbix Server 的时间相差有没有超过 sec 秒, 如果超过返回 1, 没有超过返回 0。一般对于 `system.localtime` 这个 key, 加上这个 function, 来判断 Agent 的 Linux 时间和 Zabbix Server 的时间是否相同。
- `[str|log|text].iregexp(string,sec|#num)` : 这是 `regexp` 的一个变种版本, `iregexp` 忽略了字母的大小写。i 表示 ignore, 类似 `grep -i`。
- `[floatlint|str|text|log].last(#num,time_shift)` : 最近第 #num 次数据。这里的 #num 和其他 function 中的 #num 不同, 别的 function 中的 #num 表示 “几次”, 这里的 #num 表示的是 “第几次”, 比如 `last(#3)` 表示的是离现在第三近的监控值。
- `[log].logeventid(string)` : 最近一次的日志的 Event ID 是否正则匹配 string。不匹配返回 0, 匹配返回 1。
- `[log].logseverity()` : 返回最近一次日志的等级, 返回 0 表示默认, 其他数字表示对应的等级。这个是 Windows event log 所独有的, 它可以从 Windows event log 的 “Information” 字段中抓取日志的等级。
- `[log].logsource(string)` : log 的来源和 string 是否匹配, 返回 0 表示不匹配, 返回 1 表示匹配。对 Windows Agent 比较常用。

- ◎ `[floatint].max(sec|#num,time_shift)`：最近 `sec` 秒或者 `#num` 次监控值的最大值。
- ◎ `[floatint].min(sec|#num,time_shift)`：最近 `sec` 秒或者 `#num` 次监控值的最小值。
- ◎ `any.nodata(sec)`：当在 `sec` 秒内 Zabbix Server 没有收到监控值时，返回 1，否则返回 0。
需要注意的是 `sec` 需要超过 30，否则会出现大量误报。
- ◎ `any.now()`：返回 UNIXTIMESTAMP。
- ◎ `[floatint|str|text|log].prev()`：返回上一次获得的监控值，等同于 `last(#2)`。
- ◎ `[str|log|text].regexp(string,sec|#num)`：检查最近一次的返回值是不是正则匹配 `string`。第二个参数定义的时间或者次数中的所有数据都会进行检查，只要一个符合就会认为成功。这个 function 对大小写敏感，对应的 `iregexp` 则对大小写不敏感。如果匹配成功返回 1，否则，返回 0。
- ◎ `[str|log|text].str(string,sec|#num)`，在最近一次监控值中检查是否包含 `string`。如果第二个参数不为空，那么会处理多个数据。返回值 1 表示包含，0 表示不包含。
- ◎ `[str|log|text].strlen(sec|#num,time_shift)`，最近一次获取到的监控值的长度。
- ◎ `[floatint].sum(sec|#num,time_shift)`，`sec` 秒或者 `#num` 次获取监控值的和。
- ◎ `any.time()`，返回现在的时间，格式为 HHMMSS，形如“123055”。

6.1.4 Trigger 依赖

有时候，一个 Host 的状态是依赖于其他 Host 的。比如一个交换机和它后面的一台服务器，当这台交换机出问题的时候，这台服务器自然而然的也是不可用的状态。如果对这两个 Host 都配置了 Trigger，那么当交换机出问题的时候，运维工程师会发现有两个 host 都出问题了，但真实情况是，交换机是问题的源头。

因为这种依赖性的存在，Zabbix 在 Trigger 上设计了“Trigger 依赖”。使用了“Trigger 依赖”以后，只会收到源头出问题的报警，而不是整个报警链路上的所有报警。如果一个交换机下连接着 50 台服务器，当交换机出问题时，要是没有使用“Trigger 依赖”，将会收到 51 个报警，这时，运维工程师很难找到 root cause；使用“Trigger 依赖”，只会收到一个报警，一下子就能找到问题在哪里。

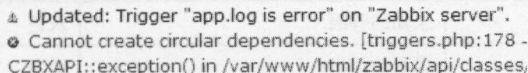
在 Zabbix 中，Host 之间是不能直接有依赖关系的，只能在 Trigger 之间有依赖关系。一个 Trigger 可以有多个 Trigger 依赖于它。

把交换机的 Hostname 定义为 SWITCH-A，后面的两台服务器的 Hostname 定义为 HOST-A，HOST-B，然后在 HOST-A 和 HOST-B 的 Trigger 上设置依赖于 SWITCH-A 的 trigger 即可。这样，

只有在 SWITCH-A 的 Trigger 变为 PROBLEM 状态时, HOST-A 和 HOST-B 的相应 Trigger 才会变为 PROBLEM, 并且 HOST-A 和 HOST-B 不会触发任何 action。

如果 SWITCH-A、HOST-A 和 HOST-B 同时出问题, Zabbix 不会触发 HOST-A 和 HOST-B 的 action。

Trigger 依赖可以在任意的两个 Trigger 之间产生, 只要不形成回路即可, 即 TRIGGER-A 依赖于 TRIGGER-B, 然后 TRIGGER-B 又依赖于 TRIGGER-A。在前端如果试图这样配置一个回路的依赖, 会报错, 如图 6-1 所示。



```

Updated: Trigger "app.log is error" on "Zabbix server".
Cannot create circular dependencies. [triggers.php:178 -
CZBXAPI::exception() in /var/www/html/zabbix/api/classes,

```

图6-1

Trigger 依赖可以从一个 Template 的 Trigger 依赖于另一个 Template 的 Trigger。当 TEMPLATE-A 的一个 Trigger 依赖于 TEMPLATE-B 的一个 Trigger 时, TEMPLATE-A 必须和 TEMPLATE-B 同时关联一个 Host, 不允许 TEMPLATE-A 单独关联, 而 TEMPLATE-B 则可以单独关联到一个 Host 上。A 依赖于 B, 所以 A 是不能脱离 B 而存在的。简单来说, 就是 Trigger 依赖可以从一个 Template Trigger 到一个 Host Trigger。比如 TEMPLATE-A 上的 TRIGGER-A 依赖于 HOST-A 上的 TRIGGER-B, 这样, 当一个 HOST-B 连接到 TEMPLATE-A 时, HOST-B 上的 TRIGGER-A 也会依赖于 HOST-A 上的 TRIGGER-B。结合前面的“交换机—服务器”例子, 可以添加一个 Template, 里面有服务器可用性 Trigger 依赖于交换机的可用性 Trigger, 那么把一批服务器加入 Template 后, 相当于这些服务器的每一个可用性 Trigger 都依赖于交换机可用性 Trigger。

有一种 Trigger 依赖是不行的, 就是从 Host Trigger 依赖于一个 Template Trigger。

Trigger 依赖的配置十分简单, 在 Trigger 的配置窗口中, 先选择“New dependency”, 再选择需要依赖的 Trigger 就可以了。

现在, 来看一下链式依赖, 还是“交换机—服务器”的例子, 这次我们在交换机前面再加一个交换机, 即网络拓扑为“交换机 A—交换机 B—服务器 A”, 如果交换机 A 出问题的话, 交换机 B 和服务器 A 都会出问题。这样的话我们需要两个 Trigger 依赖。

(1) 服务器 A 可用性 Trigger 依赖于交换机 B 可用性 Trigger。

(2) 交换机 B 可用性 Trigger 依赖于交换机 B 可用性 Trigger。

当需要恢复服务器 A 可用性 Trigger 时（即从 PROBLEM 状态变为 OK），一定要交换机 A 和交换机 B 的可用性 Trigger 都是 OK 状态才行。

6.1.5 Trigger等级

Trigger 等级，是用来定义 Trigger 的严重程度的。Zabbix 支持如表 6-3 所示的 Trigger 等级。

表 6-3

等 级	定 义	颜 色
Not classified	未知的等级	灰色
Information	提示的信息	绿色
Warning	警告的信息	黄色
Average	一般的问题	橙色
High	重要的问题	红色
Disaster	灾难性的问题	鲜红色

使用 Trigger 等级主要有以下用途。

- （1）使用不同颜色区分不同等级的 Trigger，从前端一眼就能看到问题有多严重。
- （2）不同等级的 Trigger，会有不同的警报声音。
- （3）不同等级的 Trigger 所触发的 Action，会把报警发给不同的人。比如 Information 的 Trigger，把报警发给工程师就可以了，而 Disaster 的问题，可能就要通知到领导了。

Zabbix 还可以自定义 Trigger 的等级和颜色，可以在“Administration”→“General”→“Trigger severities”定义。如果在 Zabbix 前端使用了多语言的功能，Trigger 等级名称会覆盖默认的名字。默认的 Trigger 等级名字对于所有语言都是一样的，即无论选择的是英语、汉语还是法语，看到的 Trigger 等级名称都是提到的几个。如果要根据自己的语言修改 Trigger 名称，需要对语言配置文件做一下修改。

例如，如果想把“Important”改为“重要”，那么要进入 PHP 前端目录下的 /locale/zh_CN/LC_MESSAGES/ 目录，找到 frontend.po，增加一个对应关系：

```
msgid "Important"
msgstr "重要"
```

然后保存就行了。这里可以把 msgid 和 msgstr 看成是翻译的一个键值对。

6.1.6 单位

如果使用“86400”去表示一天的秒数,是很麻烦的,一是位数多,二是容易写错。在 Zabbix 中,可以使用一些后缀缩写,来简化这个过程。主要的后缀缩写是时间单位后缀和容量单位后缀。

时间单位后缀有:s、m、h、d和w,分别表示秒、分钟、小时、天和周,可以将它们使用在 Trigger Expression、Internal Check 和聚合类型的 Item 中。

容量单位后缀有:K、M、G和T,对应计算机中的容量换算,其中 Bps 是按照 1000 来换算的,其他都是按照 1024 来换算。除了 K、M、G、T,还支持 P、E、Z、Y,每个单位都是前面一个的 1024 倍。

例如,{host:system.uptime[].last(0)}<86400,可以表示为 {host:system.uptime[].last(0)}<1d,这样是不是简单多了呢?

6.2 Events

Events(事件)是 Zabbix 重要的概念,任何一个动作的产生,都会生成一个 Event。在 Zabbix 中,一共有下面几种 Event。

(1) Trigger Event: Trigger 状态变化,会生成一个 Event。这里需要注意的是,只要 Trigger 状态变化,一定会生成 Event。即无论是 OK 变到 PROBLEM,还是 PROBLEM 变到 OK,都会生成一个 Event。Trigger 是 Event 最大的来源。

(2) Discovery Event: 当 Hosts 和 Services 被侦测到的时候产生。Zabbix 每隔一段时间会扫描一定范围(范围由网络侦测规则设定)的 IP,每当扫描到 Host 或者 Service 时,一个 Discovery Event 就产生了。具体如表 6-4 所示。

表6-4

事 件	产生的时间
Service Up	当 Zabbix 侦测到服务启动
Service Down	当 Zabbix 侦测到服务停止
Host Up	IP 至少有一个服务
Host Down	IP 上一个服务也没有
Service Discovered	第一次侦测到服务,或者服务挂掉后恢复

续 表

事 件	产生的时间
Service Lost	服务侦测到后丢失
Host Discovered	第一次侦测到服务器，或者服务器挂掉后恢复
Host Lost	服务器侦测到后丢失

（3）Auto Registration Event：当 Agent 被 Server 自动发现并且注册的时候。

（4）Internal Events：当 Item 或者是 low-level 发现规则的状态发生变化和一个 Trigger 的状态发生变化的时候产生。Internal Events 是从 Zabbix 2.2 开始的，增加这个特性，是为了用户能够了解到 Zabbix 内部的一些事件的发生，它的目标并不是发现某个 Item 出了问题，比如某个机器的磁盘满了之类，而是 Zabbix 本身可用性变化的一个事件。比如 Item 的“正常”和“异常”之间的变化，low-level 发现规则“正常”和“异常”之间的变化，Trigger 的“正常”和“未知”之间的变化。

查看 Zabbix 生成的 Events，可以从菜单进入“Monitoring”→“Events”，浏览过去生成的 Events 和它们的细节。

6.3 Actions

Zabbix 是一个监控系统，是检测数据，发出报警的，即一旦系统出现问题，要将这个“事件”告诉工程师。Zabbix 在这一方面有非常丰富的功能，当然，随之而来的就是较为复杂的设置，下面主要来了解 Zabbix 支持的报警途径，以及如何设置报警——何时报警，报警给谁。

通常，一个报警的产生，是这样的一个过程。

如果某种条件符合，那么报警。

抽象成计算机语言，就是：

```
if (ConditionA == true) {  
    Alert();  
}
```

还可以选择给谁报警（哪个用户）、怎样报警（哪种途径，短信或者邮件等），具体如下：

```
if (ConditionA == true) {
```

```
Alert (userA.email);
Alert (userBsms);
}
```

如果出了问题不一定要报警，可以在服务器上运行一些命令自动恢复，比如 Nginx 挂了，自己启动就行了，则又变成了：

```
if (ConditionA == true) {
Alert (email);
Alert (sms);
Execute (command);
}
```

再扩展一些，可能条件非常的复杂：

```
if (ConditionA == True && ConditionB != True) {
Alert (email);
Alert (sms);
Execute (command);
}
```

这就是 Zabbix 的“报警”了，为什么这里的“报警”要加引号呢？其实，发生问题后，去执行一些命令，这已经不是狭义上的“报警”了。在 Zabbix 中，这个被定义为 Action，译为“动作”，就是当某种 Condition 符合时，会进行一些操作，这些操作就叫做 Action。Condition，译为“条件”，在 Zabbix 中，Action 的触发是需要条件的，比如属于某个 Host group 的 Host，某个 Trigger 是 PROBLEM 状态了，等等。

在 Zabbix 中，报警的途径是依附于用户的。即不能直接将一个 Action 设置为给某个邮箱发邮件，一定要设置 Action 向某个用户发送报警，发送报警的途径是邮箱，那么就会发到用户的预先设置的邮箱地址了。这个邮箱地址叫做这个用户的 Media，即联系方式。

下面来介绍 Action 的设置。

6.3.1 Action

Action 是 Zabbix 非常强大的功能，可以基于 Event 的不同状态，执行不同的操作。最常见的，就是发生报警时，将报警通过各种方式发送给对应的用户。

目前 Zabbix 支持 Action 根据下面的 Events 触发。

- ◎ Trigger Events：当 Trigger 的状态变化，即从 OK 到 PROBLEM 或从 PROBLEM 到 OK 时都会产生。
- ◎ Discovery Events：当发生网络 Discovery 的时候产生。
- ◎ Auto Registration Events：当有新的 Zabbix Agent 自动注册到 Zabbix 时产生。
- ◎ Internal Events：当 Item 变为“异常”状态或者 Trigger 变成“未知”状态时产生。

下面进入 Action 的配置。

(1) 从菜单栏的“Configuration”→“Actions”进入界面。

(2) 在“Event source”下拉框中，可以选择只显示依赖于某种源头的 Action。

(3) 单击 Action 后，可看到三个标签：“Action”、“Conditions”、“Operations”。“Action”用来定义 Action 本身的一些属性和说明；“Conditions”用来定义触发 Action 的各种条件的组合、关系；“Operations”定义的是 Action 被触发后的一些操作。

默认是建立基于 trigger event 的 action，如果需要其他的，选择对应的选项即可，如图 6-2 所示。

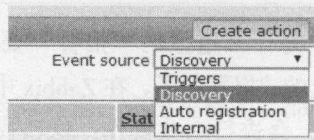


图 6-2

“Action”标签，有下面的属性可以设置。

- ◎ Name：唯一的 Action 的名字。
- ◎ Default subject：报警的默认标题。应该已经写在对话框中。
- ◎ Default message：报警的默认内容。应该已经写在对话框中。
- ◎ Recovery message：是否在问题解决后发送消息。Zabbix 将“OK”状态的 Trigger 认为是一个恢复的 recovery event。注意，如果使用了 Escalation 机制（后文有介绍，这里简单理解为报警的升级，比如，多久没处理一个报警，就要将邮件发送给老板等），Recovery event 只会触发一次。对于 Recovery 的报警，可以像发出报警的邮件一样，设置报警标题和内容。

以下几点需要注意。

- (1) 自定义的恢复信息, 只针对 Condition, 是 “Trigger value is PROBLEM” 的生效。
- (2) 恢复信息只会发送给那些之前收到过关于这个 Action 的报警信息的人。
- (3) 恢复信息和 Action 依赖的 PROBLEM 生成的 Event 维护同一份 ACK 状态。
- (4) 在 Recovery 信息中, EVENT.*Macro 中的数据, 都是基于出问题的 Event, 而不是 Recovery。

(5) 在 Recovery 信息中, EVENT.RECOVERY.* 表示的是出自 Recovery event 的数据。

- ◉ Recovery subject : Recovery 信息的标题。
- ◉ Recovery message : Recovery 信息中的内容。
- ◉ Enabled : 是否启用这个 Action。

6.3.2 Operation

Operation 指的是 Action 触发以后具体的操作, 在 Zabbix 中, 可以定义下面这些操作 :

- (1) 发送一条消息。
- (2) 执行一个命令 (包括 IPMI)。

对于 discovery 事件, 还有额外的一些操作 :

- (1) 添加一个 Host。
- (2) 移除一个 Host。
- (3) 启用一个 Host。
- (4) 禁用一个 Host。
- (5) (Host) 添加到一个 Host group。
- (6) (Host) 从一个 Host group 中删除。
- (7) 关联到一个 Template。
- (8) 取消和一个 Template 的关联。

对于 auto-registration 事件, 也有额外的一些操作 :

- (1) 添加一个 Host。

- (2) 禁用一个 Host。
- (3) 添加到一个 Host group。
- (4) 关联到一个 Template。

在配置 Action 的 Operation 标签页时，可以看到目前配置的 Operation，单击“New”按钮，如图 6-3 所示。

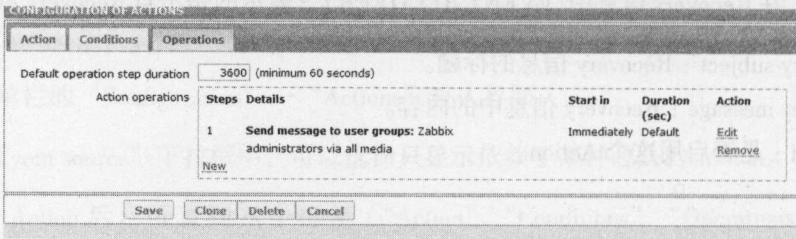


图 6-3

随后会出现配置一个新的 Operation 的界面，如图 6-4 所示。

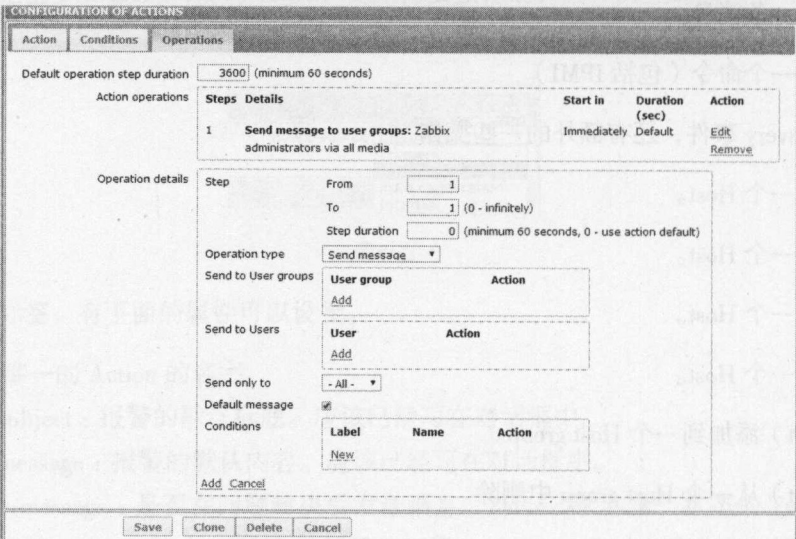


图 6-4

以下对各部分进行详细说明。

- ⊙ Default operation step duration：最小是 60 秒。若设置为 1 小时（即 3600 秒），则表明执行了一个操作后，要等待 1 小时，才会执行下一个操作。

◎ Action operations : 需要设置的有 Steps、Details、Start in、Duration(sec) 和 Action。

- Steps: 在 escalation (报警的扩散、升级) 的时候, 会按照 Step 的顺序来执行, 从 1 开始。
- Details: 操作的类型和目标。从 Zabbix2.2 开始, 还会显示在发送信息时的 media type (电子邮件、SMS、Jabber 等), 用户的名字也会显示。
- Start in : 在 Event 发生后多久执行操作。
- Duration(sec) : 显示的是 Step 的持续时间, 如果 Step 使用了默认的“持续时间”, 那么显示“Default”。
- Action : 显示的是两个标签“Edit”、“Remove”, 用来编辑和移除 Operation 的操作。

◎ Operation details : 用来设置一个 Operation 的具体设置。

下面我们看看 Operation details 的设置。

- Step : 在 Escalation 的过程中的执行计划。
- From : 表明从哪一步开始。
- To : 表明到哪一步结束。
- Step duration : 每一步持续的时间, 如果填 0, 就使用上面的“Default operation step duration”中的值。可以在同一个步骤中, 进行多个操作。如果这些操作有多个 duration, 那么会选择最短的那个生效。
- Operation type : 选择操作的类型, 可以选择的有如下两种。

Send message : 给用户发送信息 (这里的信息可以是邮件等)。

Remote command : 远程执行命令。

注意: 对于 discovery 事件和 auto-registration 事件, 可以在这里选择更多的操作。

Operation 具体执行的操作, 是 Operation 的核心。在 Zabbix 中, “Send message” 和 “Remote command” 是最重要的两个 Operation。报警的核心是什么? 其一, 是将问题通知到负责人; 其二, 是对报警有应急的措施。前者对应的是 “Send message” 的功能, 后者是 “Remote command” 的功能。比方说, 当一台 PHP 服务器的 PHP 进程意外退出后, Zabbix 一边发送邮件给负责人, 一边向 Zabbix Agent 发出命令, 让它重启 PHP 服务器上的 PHP 进程。这是非常棒的报警处理流程。下面, 一起看下如何订制 Operation。

首先看 “Send message”, 这个应该是所有 Action 都会具备的 Operation。就算能够自动恢复 (比如 PHP 进程重启), 也需要将出错的信息及时发送给负责人。对于 “Send message” 的配置有如下几点。

- Send to User groups : 可以添加一些 User group (犹如 Host group 之于 Host 的关系, User group 就是一组用户), 将报警发送给 User group 中的所有 User。
- Send to Users : 类似于 “Send to User groups”, 只是发送警报的对象换成用户。
- Send only to : 选择的是给 “Send to User groups” 和 “Send to Users” 中发送消息时使用的 Media type。比如选择了 “Email”, 那么就会向前面的 User 发送电子邮件。
- Default message : 使用默认的消息格式。默认这个是被打上勾的, 取消选择, 可以看到默认定义的消息格式。
- Conditions : 在后面进行介绍, 因为它的设置是 “Send message” 和 “Remote command” 所共有的。

注意, 对于一个 Host 的报警, Zabbix 只会把这个报警发送给对这个 Host 至少有 “读” 权限的用户。Trigger 中至少一个表达式关联的 Host 是正常工作的, 即在 Host 中看到是绿色的标识, 如图 6-5 所示左侧的 “Z”。

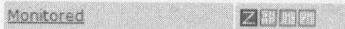


图6-5

对于发送出去的消息, 怎么查看历史消息呢? 怎么获知什么时间发送了什么消息呢? 在 Monitoring-Events 中可以看到有触发的 Action 的列表。红色表示 Action 是失败的; “In progress” 表示 Action 已经被处罚了; “Failed” 表示没有 Action 触发成功。

我们单击 Event 的时间, 可以看到 Action 的细节, 包括发送了信息的具体内容。

同时, 我们也可以通过 “Administration” → “Audit”, 在过滤条件中选择 “Action”, 就可以看到过去一段时间内发生的所有 Action。

“Remote command” 的参数有以下几种。

- Target list : 选择命令执行的 Host, 可以选择发生问题的 Host, 指定某个 Host 或者 Host group。
- Type : 选择执行的命令的类型, 其中 “IPMI”、“SSH”、“Telnet” 很好理解, 主要看剩下的两个。
 - Custom script : 执行在 “Commands” 对话框中的 shell 命令。
 - Global script : 在 Administration-Scripts 中定义的一些命令。
- Execute on : 可以选择在 Zabbix agent 还是 Zabbix server 上运行命令。
- Conditions : 后面进行单独介绍。

Remote command 最大的好处是什么呢？是自动。Zabbix 会根据配置的条件，去执行对应的命令，下面看看 Remote command 的应用场景。

- (1) 应用无法响应时，自动重启某些应用。
- (2) 当在服务器不响应时，使用 IPMI 的 “reboot” 命令重启服务器。
- (3) 在磁盘要满了的情况下，自动删除一些文件（比如 /tmp）。
- (4) 根据 CPU 负载，自动进行虚拟机的调配。
- (5) 弹性计算，根据系统情况，新增或删除云节点。

Zabbix 无法通过 Zabbix Proxy 向 Zabbix Agent 发送，一定要从 Zabbix Server 发起。而且，发送的命令长度也有限制，即不能超过 255 个字符，这个对于一般的命令是绰绰有余的了，只要不是 cat 个文件之类的，都足够了。如果在多行写多个命令，Zabbix 会按照顺序执行。而且，在 Remote command 中，还支持 Macro 定义。

相比上面介绍的发送消息，Remote command 稍显复杂。在 Agent 上执行的自定义脚本（即 Custom scripts）一定要在 zabbix_agentd.conf 中预先定义，而且在 zabbix_agentd.conf 中，“EnableRemoteCommands” 这一项要设置为 1，否则无法远程执行命令。修改后记得重启 Zabbix Agent。还有一点要注意，对于 active 模式的 Zabbix Agent，是无法执行远程命令的。这个是必然的，因为 active 模式的 Zabbix Agent 其实根本没有在服务器上安装 Zabbix Agent，怎么能发送命令给它让它执行呢？

对于远程执行命令，权限也是个问题。默认情况下，Zabbix 是没有权限来重启系统服务的，如果 Zabbix 用户想要有这个权限，需要修改下 sudoer 文件。

```
shell> visudo
# 允许“zabbix”用户不需要密码就可以运行所有 root 权限的命令
zabbix ALL=NOPASSWD: ALL
# 允许“zabbix”用户可以在不需要密码的情况下运行 /etc/init.d/apache restart，即重启 apache
zabbix ALL=NOPASSWD: /etc/init.d/apache restart
```

如果 Host 上某一类 Interface 有多个（比如多个 Zabbix Agent 实例），那么 Zabbix 会选择默认的去运行。

对于剩下的“Conditions”，它有两个选项“Not ack”和“Ack”，“ack”是“acknowledge”

的缩写，在 Zabbix 中，意为某个 Event 是否被人“认领”了，可以理解为，有没有人在处理这个事情。这里的“Not ack”和“Ack”，表达的是在何种情况下需要执行 Operation。如果选择“Not ack”，那么只有当 Event 没有被“Ack”的情况下需要执行。

6.3.3 Condition

报警，肯定是基于某个条件的，比如某个服务器的 CPU 负载超过 20。在 Zabbix，这种“条件”就是 Trigger，那不能对每一个 Trigger 都设置一个 Action 吧？最好的办法就是定义某一类的 Trigger 如果出问题了，就统一触发某个 Action。Zabbix 就是这么做的，它在 Trigger 和 Action 之间，抽象了一个 Condition 的概念。“Condition”的中文意思是“情况”，可以理解为某一种条件。即 Action 不是直接和 Trigger 挂钩的，而是可以配置一组条件，如果都满足这些条件，就执行 Action。比如 CPU 负载超过 20 这个 Trigger，可能对于消耗 CPU 的服务器来说不需要报警，但对于不消耗 CPU 的服务器来说需要，那么可以组合这两个条件“CPU 负载超过 20”和“服务器是 CPU 密集型”，对应到 Zabbix，就是“CPU>20”且“Host 属于 CPU Host group”。

最常用的是基于 Trigger 的 Event，在表 6-5 中，提到的 Host 等，指的都是和这个 Event 相关的 Trigger 中关联的 Host。

表 6-5

Condition 类型	支持的操作	说 明
Application	=, like, not like	限定 Application，我们把生成这个 Event 的 Trigger 所关联 Item 的 Application 称为 AppA，设置的字符串称为 APP = : AppA 的名字和 APP 完全一致 like : AppA 的名字中包含“APP” not like : AppA 的名字中不包含“APP”
Host group	=, <>	Host 是否属于一个 Host group
Template	=, <>	Trigger 是否属于一个 Template
Host	=, <>	Host 是否是某一个 Host
Trigger	=, <>	触发的 Trigger 是否是某一个 Trigger
Trigger name	like, not like	Trigger 名字是否和一个字符串匹配
Trigger severity	=, <>, <=, >=	Trigger 的严重等级的范围
Trigger value	=	Trigger 是 OK 还是 PROBLEM
Time period	in, not in	Event 生成的时间是否属于某一个范围

续 表

Condition 类型	支持的操作	说 明
Maintenance status	in, not in	Host 是否在 Maintenance 状态。如果 Trigger 中有多个 Host, 至少其中一个是 (或者不是) Maintenance 状态

如果设置的 Condition 中的任何一个对象 (Host 等) 被删除了, 那么这个相关的 Condition 会被删除, 这个 Action 也会被禁用, 防止出现错误执行 Action, 并且只能由用户自己重新启用。

Trigger 的值是会变的, 如果设置了 “Trigger=Problem”, 表示的是当 Trigger 从 OK 变成 PROBLEM 的时候会触发。反之亦然。

当创建一个 Action 的时候, 默认会有两个 Condition: 一个是 “Trigger=PROBLEM”, 另一个是 “Maintenance status=not in maintenance”。为什么 Zabbix 要有这两个 Condition 呢? 其实仔细想想很有道理, 一般来说, 我们的 Action 都是在某样东西出问题时才需要行动的, 而且, 这个东西还不能是在维护中, 否则明明有人是在维护这台服务器, Zabbix 却还在拼命报警, 就不好了。

表 6-6 是基于 Discovery 的 Event 可以使用的 Condition。

表 6-6

Host IP	=, <>	IP 是否在某个范围内
Service type	=, <>	Discovery 是否属于某个服务, 服务有 SSH 等, 就是在创建 Discovery 规则时可以选择的那几个
Service port	=, <>	服务的端口是否在某个范围内
Discovery rule (Discovery check)	=, <>	Discovery 规则是否是某一个特定的
Discovery status	=	Discovery 属于哪种状态, 共有 Up, Down, Discovered 和 Lost 四种
Uptime/Downtime	>=, <=	Up 状态或者 Down 状态超过或者小于一个时间段
Receiver value	=, <>, >=, <=, like, not like	收到的数据是否满足一定条件
Proxy	=, <>	是否使用某个 Proxy 监控

表 6-7 是基于 Active agent auto-registration 的 Condition。

表 6-7

Host metadata	like, not like	Host 的元数据是否满足条件
Host name	like, not like	Host 的 Hostname 是否满足条件
Proxy	=, <>	是否被某个 Proxy 监控

表 6-8 是基于 Zabbix 内部事件的 Condition。

表 6-8		
Application	=, like, not like	和基于 trigger 的 event 中的介绍一致
Event type	=	事件的类型（目前支持的类型我们在表格后介绍）
Host group	=, <>	Host group 是否是某一个 Host group
Template	=, <>	Template 是否是某一个特定的
Host	=, <>	Host 是否是某一个 Host
Node	=, <>	Event 是否属于某一个 Node（Node 是 Zabbix 分布式部署的一个节点）

Event type 中的事件类型有以下几种。

- Item 是 “not supported” 状态。
- Item 是 “normal” 状态。
- Low-level discovery 规则是 “not supported” 状态。
- Low-level discovery 规则是 “normal” 状态。
- Trigger 是 “unknown” 状态。
- Trigger 是 “normal” 状态。

Event 支持的 Condition 都介绍完了，对于不同的 Condition 的组合，Zabbix 也有一套逻辑，比如需要同时满足两个 Condition，又或者只要满足两个 Condition 中的一个等，Zabbix 支持的 Condition 之间的逻辑运算符有以下几种。

- AND：所有 Condition 同时满足。
- OR：所有 Condition 满足一个就行了。
- AND/OR：根据选择的条件，自动调整。选择相同类型的 Condition 时，它就变成 and；如果选择不同的 Condition，它就变成 OR。比如有下面这些 Condition：
 - Host group = Oracle servers
 - Host group = MySQL servers
 - Trigger name like 'Database is down'
 - Trigger name like 'Database is unavailable'

那么最后组合的 Condition 就是（Host group = Oracle servers or Host group = MySQL servers）and（Trigger name like 'Database is down' or Trigger name like 'Database is unavailable'）

6.3.4 Escalations

Escalation 的意思是“增加, 扩大”, 在 Zabbix 中, 它指的是一个报警在一定条件下, 会执行一些额外的操作。比方, 一台服务器磁盘满了, 可能马上需要通知的是一线的运维工程师。如果 6 个小时都没有人处理, 这个故障还没有恢复, 那么可能就要汇报给经理了。或者, PHP 进程挂起了, 可能首先是重启 PHP 进程, 那么如果过了一段时间这个故障还没有恢复 (即 PHP 进程没有重启成功), 那么就要通知工程师来进行恢复了。这是一个报警扩散的过程, 即 Escalation。

Zabbix 中, 支持的 Escalation 有以下几种。

- 发生问题后, 第一时间通知用户。
- 在问题解决前, 每隔一段时间向用户报警。
- 延迟报警。
- 报警可以升级, 发送给更多的用户。
- Remote command 可以在事件发生后马上执行, 也可以在一定时间没有解决后才执行。
- 可以向用户发送恢复通知。

可以定义一个“Escalation Step”, 意为“扩散步骤”, 定义何时扩散报警, 以及如何扩散。每一个步骤可以定义一个 Action 和持续时间。步骤 (1) 要在报警发生后马上发出, 步骤个数没有限制, Zabbix 只会从第一个开始逐个执行。

Escalation 是一个比较复杂的机制, 特别是跟其他的东西结合起来之后, 下面看一些常见情况。

(1) 出问题的 Host 在发出第一个报警后进入了 Maintenance 状态: 这个 Action 剩余的 Escalation Step 都会被执行。Maintenance 状态不会停止 Operation, 只会对 Action 有关系。简单地说, 一旦这个 Action 被执行, 那么其中的每一步都会执行。

(2) 在 Time period 中定义的时间在发出第一个报警后就结束了: 同 (1) 中的情形, Time period 也只会影响 Action 执行与否, 而不会影响 Action 中的 Operation 执行与否。

(3) 在 Maintenance 状态时发生了问题, 并且在 Maintenance 状态结束后依然没有恢复: 所有 Escalation Steps 都是从 Host (或者其他) 结束 Maintenance 状态后开始。

(4) 当 Host 在 no-data Maintenance 状态时发生问题, 在结束 no-data Maintenance 状态时, 这个问题还没有恢复: Trigger 的触发, 一定是先于 Escalation Steps 的开始的。

(5) 不同的 Escalation Steps 非常接近互相有重叠部分: 每一个 Escalation 都会接替之前的

Escalation，但是由于步骤（1）是在问题发生后马上执行的，所以“之前的 Escalation”至少会执行一个动作。这些行为跟 Event 和 Action 相关。

（6）在 Escalation 执行过程中，Action 被禁用了：正在发送过程中的信息和之后的那一条信息会被发送。其中后面的那条信息会在发送的信息前加上“NOTE: Escalation cancelled: action ‘<Action name>’ disabled”。这样，用户就会知道 Action 已经被禁用了，之后也不会收到关于这个 Action 的消息了。

Escalation 稍微有点复杂，但很有用，下面一起看几则关于 Escalation 的例子，希望能帮助大家理解。

1. 示例 1

要求每隔 30 分钟向“MySQL Administration”的 User group 发送一次报警，一共发送 5 次。

（1）在 Action 的 Operation 标签中，将“Default operation step duration”（默认操作间隔时间）设置为“1800”秒，即要求中的 30 分钟。

（2）在 Steps 的地方设置为“From 1 to 5”，表示 Escalation Step 的第一到第五步都是执行这个操作。

（3）选择“MySQL Administration”组作为发送报警的收件人。

通过这样的设置，假设 Action 是 0 点 0 分触发的，那么在 0 点 30 分、1 点、1 点 30、2 点都会将报警发送给 MySQL Administration 用户组中的所有用户，当然，如果在这个过程中，Trigger 恢复了，那么就会打断这些事件。

2. 示例 2

如果示例 1 中的问题一直没有解决，我们希望把这个问题通知到更加资深的 DBA，可以进行下面的设置。

（1）在 Operation 标签中，将默认时间设置为“36000”秒，即 10 个小时。

（2）将 escalation steps 设置为“From 2 to 2”，意思就是只在第（2）步中执行。

在问题发生后，如果 10 个小时还没有恢复，那么这个问题就会通知到资深 DBA，可以在发送消息的内容中加上类似“这个问题已经 10 小时没有处理”之类的话，提醒收到报警的工程师去解决。

3. 示例 3

当出现问题时，先通知 MySQL Administration，如果问题持续 10 个小时，将这个问题发送给 DBA 经理，如果还解决不了，会尝试重启数据库。如果依然解决不了，那么只能发邮件通知用户，最后使用 IPMI 命令，重启 MySQL 服务器。如图 6-6 所示。

Action	Conditions	Operations		
Default operation step duration 1800 (minimum 60 seconds)				
Action operations				
Steps	Details	Start in	Duration (sec)	Action
1 - 0	Send message to user groups: MySQL Administrators via Email	Immediately	Default	Edit Remove
5	Send message to user groups: Database manager via Email	02:00:00	Default	Edit Remove
6	Run remote commands on current host	02:30:00	Default	Edit Remove
7	Send message to user groups: Guests via Email	03:00:00	Default	Edit Remove
9	Run remote commands on current host	04:00:00	Default	Edit Remove
New				

图 6-6

4. 示例 4

最后看一个自定义 Duration 的例子，先看是如何设置 Action 的，如图 6-7 所示。

Action	Conditions	Operations		
Default operation step duration <input type="text" value="1800"/> (minimum 60 seconds)				
Action operations				
Steps	Details	Start in	Duration (sec)	Action
1 - 4	Send message to user groups: MySQL Administrators via Email	Immediately	Default	Edit Remove
5 - 6	Send message to user groups: Database manager via Email	02:00:00	3600	Edit Remove
5 - 7	Send message to user groups: Zabbix administrators via Email	02:00:00	600	Edit Remove
11	Send message to user groups: Guests via Email	04:00:00	Default	Edit Remove
New				

图 6-7

假设问题是在 00:00 发生的，那么它的执行顺序如下。

- (1) 在 00:00、00:30、01:00、01:30 会向 MySQL administration 用户组发送邮件，这是由于我们设置了默认的时间间隔是 1800 秒，即 30 分钟。
- (2) 在 02:00 和 02:10 向 Database manager 用户组发送邮件。
- (3) 在 02:00、02:10 和 02:20 向 Zabbix administration 用户组发送邮件。
- (4) 在 04:00 向 Guests 用户组发送邮件。

这里有几个理解起来比较麻烦的地方，一个是在图 6-7 的 2 中，只有在 02:00 和 02:10 时才会向 Database manager 用户组发送邮件，而不会在 03:00，这是因为在图 6-7 的 5-6、5-7 都设置了 Operation，那么在 5-7 中设置的 600 秒就会覆盖 5-6 中设置的 3600 秒。在 3 中，因为设置的 600 秒生效，所以每隔 10 分钟向 Zabbix administration 发送一次邮件。在 11 中，由于经过了 8,9,10,11 这 4 个 step，所以是默认的 30 分钟的 4 倍，即 2 个小时，到 04:00，向 Guests 发送报警。

6.3.5 Unsupported状态的Items的报警

Internal events（内部事件）是 Zabbix 的新概念，它不同于其他的 event，它的生成，是由于 Zabbix 内部的东西出现了问题。Internal events 会在下面几种情况下生成。

（1）Items 从“normal”状态变成“unsupported”状态，或者从“unsupported”状态变成“normal”状态。

（2）Triggers 从“normal”状态变成“unknown”状态，或者从“unknown”状态变成“normal”状态。

（3）Low-level discovery 规则从“normal”状态变成“unsupported”状态，或者从“unsupported”状态变成“normal”状态。

对于这些 Internal events 的报警，设置起来非常简单，但设置的方法有些隐蔽。

首先在设置 Action 界面右上角的“Event source”下拉列表中，选择“Internal”选项。如图 6-8 所示。

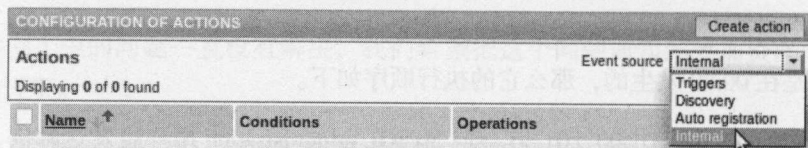


图6-8

然后在配置 Condition 的时候，选择“Event type”选项，就可以配置 internal events 的报警了，如图 6-9 所示。

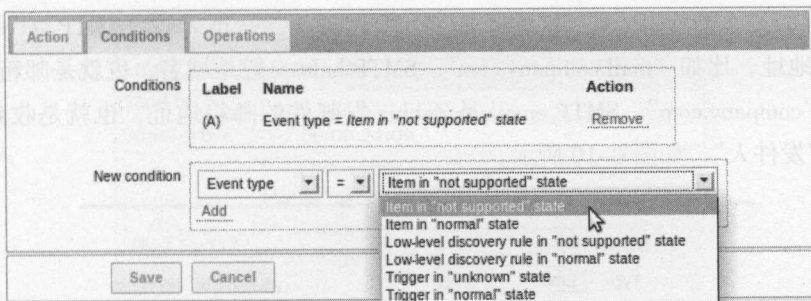


图6-9

监控 unsupported 的 Items 的意义是什么呢？使用 Zabbix 1.8 时，监控 Zabbix 本身是个很麻烦的事情，当时是使用 Nagios 来监控 Zabbix 的，但用一个监控系统监控另一个系统明显是有漏洞的。对于不支持的 Items，我们是写了个 SQL 从 Zabbix 数据库中抓取 unsupported 的 Items 数量，作为一个衡量 Zabbix 监控质量的指标。而到了 Zabbix 2.2，这些都已经内置的了，对于使用维护 Zabbix 的工程师来说，轻松了不少。

6.4 Media类型

Zabbix 支持的 Media 一共有以下几种。

- (1) 电子邮件。
- (2) SMS。
- (3) Jabber。
- (4) Ez Texting。
- (5) 自定义报警脚本。

下面分别介绍它们的用途和设置。

1. 电子邮件

要让 Zabbix 能够发邮件，需要设置 SMTP 相关的参数。

首先需要配置 Zabbix 发送邮件的服务。进入“Administration”→“Media types”中，新建一个 Media 或者单击原有的 Email。

需要配置的是 SMTP server、SMTP helo 和 SMTP email。一般来说，SMTP server 输入的是 SMTP 服务器地址，比如“mail.company.com”。SMTP helo 一般是域名，也就是邮箱“@”后面的地址，如“company.com”。SMTP email 是 Zabbix 发邮件的邮箱地址，也就是收到 Zabbix 报警邮件时的“发件人”，如图 6-10 所示。

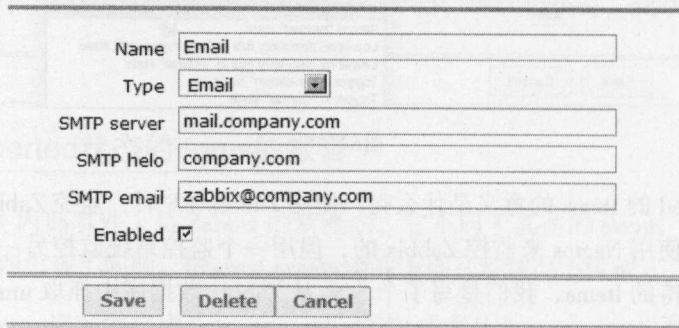


图 6-10

现在就可以给用户设置邮箱了，从菜单栏进入“Administration”→“Users”，在右上角选择 Users，如图 6-11 所示。

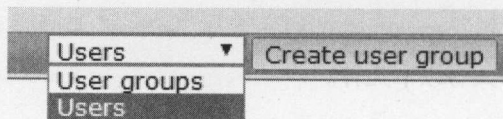


图 6-11

进入用户配置界面后选择 Media 标签页，如图 6-12 所示。

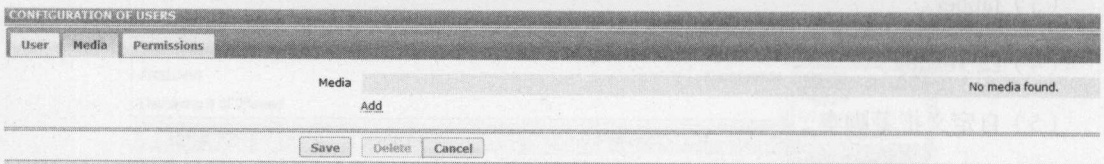


图 6-12

单击“Add”按钮后会出现配置的地方，如图 6-13 所示。

图6-13

其中,

- Type : 类型, 比如需要设置的 Email。
- Send to : 电子邮箱的地址, Zabbix 会把邮件发送到的邮箱。
- When active : 可以对这个邮箱设置启用的时间, 比如在公司时使用公司的邮箱, 下班时使用另一个邮箱。
- Use if severity: 和 Trigger 有关, 其中包括的“Not classified”等, 都是 Trigger 的严重等级。这个选项的意思就是只有与勾选的严重等级对应的 Trigger 触发时, 才会发邮件给这个邮箱。
- Status : 选择启用, 还是不启用。

2. SMS

Zabbix 支持向手机发送短信报警, 但是是需要硬件支持的——一个 GSM 调制解调器, 这是个挺麻烦的事情。据我了解, 国内还没有使用这个东西来实现 Zabbix 短信报警。因此下面这些内容, 参照了 Zabbix 官方文档对 SMS 的说明, 我本人没有实践过, 如有疏漏, 敬请包涵。后文介绍了微信报警的功能, 也可作为替代方案。

在使用 GSM 调制解调器之前, 首先要确认以下几点。

(1) 电脑串口的速度 (在 Linux 下一般是 /dev/ttyS0) 和 GSM 调制解调器的速度相同。Zabbix 不会设置串行连接的速度, 只会使用默认值。

(2) 运行 Zabbix 的用户, 要有这个串行设备的读写权限。

（3）GSM 调制解调器输入 PIN 后，需要重新启动，也可以在 SIM 卡上屏蔽 PIN 保护功能。PIN 可以在终端（Linux 的 Minicom 或者是 Windows 的 HyperTerminal）输入：AT+CPIN="NNNN"，"NNNN" 是 SIM 卡的 PIN，这里的双引号一定要有。

Zabbix 在 Siemens MC35 和 Teltonika ModemCOM/G10 这两个 SMS 调制解调器上测试过。

有了以上这些的硬件配置，还需要在 Zabbix 的 Media 中配置这个 SMS 服务，并且给 User 添加对应的联系方式。

从 "Administration" → "Media types" 进入，可以创建一个 Media，也可以修改 Zabbix 原有的 SMS。它有个参数可以如下配置。

- Description：这个 Media 的描述。
- Type：这里选择 SMS。
- GSM modem：选择 GSM 调制解调器的串行设备名字。

在添加 User media 的时候，在 Send to 这个属性中，填写需要发送短信的手机号码就行了。

3. Jabber

Jabber 中国人用的也比较少，简单来说，Jabber 就是一种通信协议，可以用它创建聊天室。而 Zabbix 使用 Jabber，就跟发电子邮件一样，把报警（或者消息）通过 Jabber 协议发送出去。

要配置 Jabber 其实很简单，进入 Jabber 的配置页，唯一不同的就是参数 "Jabber identifier" 和 "Password"，照着意思输入就行了。

在 "User media" 中，在 "Send to" 中输入需要发送的 Jabber 地址。

需要注意的是，Zabbix 会优先选择从 Jabber SRV 记录解析域名，如果失败了，再从地址记录中寻找。在 Jabber SRV 记录中，优先级最高的、权重最大的会首先选择。

说明：

（1）SRV 记录是在 DNS 中的特殊数据，用来定义位置、Hostname 和端口之类的信息。比如：

```
_sip._tcp.example.com. 86400 IN SRV 0 5 5060 sipserver.example.com
```

表示的就是 sipserver.example.com 监听在 5060 的 TCP 端口，它的优先级是 0，权重是 5。

（2）Address 记录一般被称为 "A 记录"，用来指定域名（或者主机名）对应的 IP。

4. Ez Texting

Ez Texting 也是一个国内非常少人使用的东西，简单来说，它是一个发送 SMS 的服务商，就把它简单理解为 QQ 好了，在 Ez Texting 登录后，选择要发送的号码，就可以把 SMS 信息发送出去了。

需要配置的也很简单，可以按字面意思理解。

○ Username：用来登录 Ez Texting 的用户名。

○ Password：用来登录 Ez Texting 的密码。

然后在“User media”页面上，在“Send to”中填写收件人的地址。

5. 自定义报警脚本

如果前面的几种发送报警的方式（其实主要就是邮件）都不能让你满意，那么可以自己定义发送报警的脚本。这些脚本必须在 Zabbix Server 的配置文件的 AlertScriptsPath 中，当这些脚本执行的时候，需要三个参数。

(1) To：就是 User media 中的“To”一栏。

(2) Subject：标题。

(3) Message：报警的内容。

代码如下。

```
#!/bin/bash
to=$1
subject=$2
body=$3
cat <<EOF | mail -s "$subject" "$to"
$body
EOF
```

配置非常简单，在“Administration”→“Media types”中新建一个 Media type，在 Script name 中输入脚本名即可。

6.5 Maintenance状态

一台服务器在服务的过程中，有很多时间是处于维护状态的，可能是在上线，也可能是在搬迁机器，在这个过程中，有时会造成一些报警。比如：正在进行的操作会造成 CPU 负载上升；进行压力测试时，服务器不需要监控或报警；搬迁一个机房的服务器时，如果监控和报警仍在继续，会收到很多关于这个机房机器 unreachable 的报警。

在 Zabbix 中，支持这样的功能——Maintenance，它可以帮我们解决上面的问题。Maintenance 状态分为两种：一种是在状态中停止对目标的监控数据的收集，另一种是在状态中继续对目标的监控数据的收集。当不想收到维护中的服务器的报警时候，需要在 Action 中设置条件“Maintenance status = not in ‘maintenance’”。不用在每个 Action 新建时去增加这个限制，这是 Action 在建立时就默认存在的一条规则。

当 Action 的条件中有多个 Host 时，只要其中有一个 Host 是不在维护状态的，就会发送这个报警。

对于 Maintenance 状态的运行机制，这里简单介绍一下。Zabbix 有个进程叫做“Timer”，它负责设置 Host 的维护状态，在每一分钟的 0 秒，它会根据设置将 Host 的状态修改为“维护中”，又或者是将其从“维护中”去除。代理会一直收集 Host 的数据（包括设置为不收集监控数据的维护状态），这里大家可能会认为我写错了，因为前面不是说了“维护中”状态可以设置为不收集监控数据吗？事实上，代理是会收集这些我们已经设置为不收集数据的 Host 的，但在代理将数据发送给 Zabbix Server 的时候，Zabbix 会忽略这些数据。

有一点 Zabbix 设计得很好：当一个 Host 从“no data”维护状态中恢复时，是不会触发 nodata() 这个方法的，因为 nodata() 是跟上一次的结果做比较，所以不会认为是“no data”。

“维护中”状态是非常棒的一个机制，它很简单，但非常有用，下面看看如何配置。

从菜单栏的“Configuration”→“Maintenance”进入，一共有三个标签：Maintenance、Periods 和 Hosts&Groups。“Maintenance”标签中配置了一个 Maintenance 的开始和结束时间，以及名称和详细描述。在“Maintenance type”中可以定义在这段维护时间中，是否需要收集维护中 Host 的监控数据。

Periods 标签，是定义和时间相关的东西的。在“Maintenance”标签里面，只能定义比较死板的时间，从一个时间点到另一个时间点。但有时，我们有一个固定的维护时间，比如每个星期一的 10 点到 12 点，那么就可以使用 Periods 标签来实现了。

如图 6-14 所示，可以设定每天，每周和每个月的维护时间。

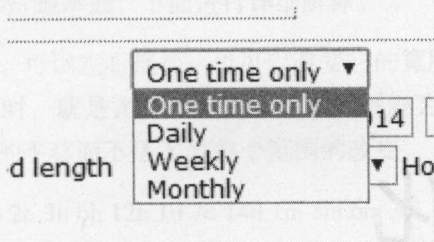


图6-14

而在 Host&Groups 标签中，我们可以设置对于哪些 Host 和 Host group 生效。

第 7 章

数据可视化

7.1 Graph

Zabbix 的数据非常多，每秒都有几百个数据流入，所以一个好用的数据可视化工具对于用户来说是非常重要的。当一台服务器出问题的时候，运维工程师最常用的方法就是看历史数据的变化趋势，比如 CPU 负载上升了，就要用 `sar -q` 看到底是从什么时候起 CPU 负载出了问题；又比如磁盘读写变高，就要用 `sar -b` 看过去一段时间内磁盘读写的变化趋势。Zabbix 对于它收集的数据，提供了 Graph 模块，如图 7-1 所示。

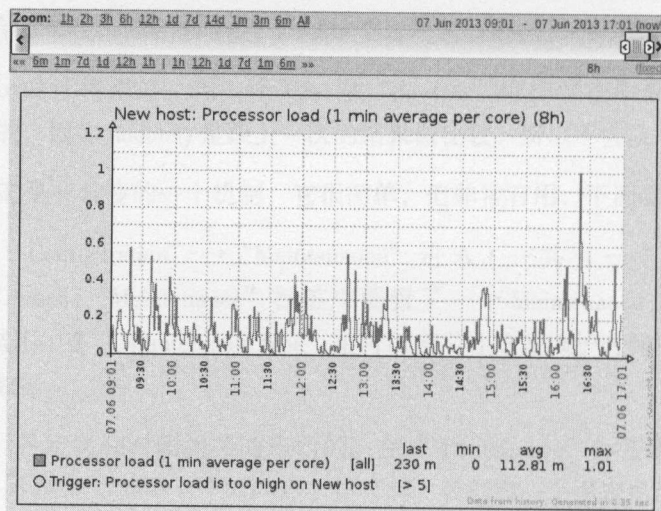


图 7-1

图 7-1 的上部分是选择时间的部分，这非常重要，很多人在熟练使用 Zabbix 之后，对于 Graph 的时间选择还不能 100% 地掌握，下面进行详细解释。

上半部分中间的滚动条，可以左右拖动，也可以改变它的宽度。它的宽度表示 Graph 中 X 轴的范围，比如上图中 8 小时，就是 X 轴的范围，也可以理解为 X 轴的宽度。左右拖动这个滚动条，表示的是时间范围的平移而不是 X 轴整个范围的变化。

左上部分的标签，即 1h 2h 3h 6h 12h 1d 14d 1m 3m 6m All，是缩放 X 轴的范围的；左下角的标签，是平移当前 X 轴的。这两个标签的作用，和直接拖动滚动条的效果是一致的。但使用标签可以进行精确的时间选择。中间的滚动条在使用时，很难能精确到我们想要的时间。滚动条两边有“<”和“>”，也能移动滚动条。

如果左边的标签还不满足需求，需要更精确地选择想要的时间，比如要精确到分钟，就要使用上半部分的右上角的工具，单击两个日期区域，会跳出日历，可以选择精确的日期和时间。

在图 7-1 中，右下角有“fixed”的字样，单击以后会变成“dynamic”，它们的区别在于：当为“fixed”时，使用左下角的标签时平移时间后，X 轴的范围不会变。比如现在的开始时间是 10 点，结束时间是 15 点，单击向左平移 1h 后，时间范围变为 9 点到 14 点。如果是 dynamic，那么会变成 9 点到 15 点。单击滚动条两侧的“<”和“>”，也不会改变 X 轴的范围。

还有个简单的方法，可以在 Graph 中深入到某一个时间，就是直接用鼠标在 Graph 上拖动。

Graph 在绘制时间跨度范围比较大的时候，会使用聚合后的数据，即 Trends，我们可以从 Graph 的右下角看到数据是从 History 还是 Trends 来的。比如图 7-1 中的“Data from history”。如果是 Trends 图，会看到三条曲线，绿色的是平均值，粉红色是最大值，浅绿色是最小值，黄色区域是这段时间内的值的范围。working time（又或者是 working days，可以在“Administration”→“General”→“Working time”中设置）的背景是白色的，非 working time 是灰色背景。如果时间范围超过 3 个月，那么 working time 不会显示。

什么时候使用 History，什么时候使用 Trends 数据呢？Zabbix 遵循下面这些规则：

（1）Item History 保存的时间。比如 Item 保存 14 天 History，那么当 X 轴的时间范围超过 14 天时，就会使用 Trends 数据。

（2）如果水平方向需要绘制的像素点超过 3600/16 个，就会使用 Trends，无论 X 轴范围是否在 History 可用的范围内。

（3）如果 Trends 被禁用了并且 X 轴的时间范围在 History 范围内，Zabbix 会使用 Item history 来构建 Graph。

在右上角的下拉框，可以选择显示 “Values/500 latest values”，这里显示的数据是罗马数据，没有单位，不会乘以系数（在 Item 中设置），Value mapping 也不生效。

自定义 Graph

前面介绍了 Zabbix 的数据可视化模块——Graph，下面一起学习如何对 Graph 的表现方式进行自定义。

最普遍的需求，就是把两个 Item 放到同一个 Graph 里面查看，比如一台服务器的网卡进流量和网卡出流量。这就需要自定义 Graph：找到对应的 Host 或者 Template，选择 Create graph。下面解释下每一个配置的用途：

- ◎ Name：唯一的 Graph 名字。Item 的值我们可以使用 {host:key.func(param)} 来表示，需要注意的是这里的 fun 仅仅支持 avg、last、max 和 min。在这个宏中，也可以使用 {HOST.HOST<1~9>} 来表示 Graph 中 Item 对应的 Host，比如：{{HOST.HOST<1>}:key.func(param)}。

- ◎ Width：Graph 的宽度。

- ◎ Height：Graph 的高度。

- ◎ Graph type：Graph 的类型，Zabbix 支持的有以下几种。

- Normal：普通的 Graph。

- Stacked：柱状图。

- Pie：饼图。

- Exploded：分裂的饼图。

- ◎ Show legend：是否显示 legend，legend 会告诉你某条曲线的名称。

- ◎ Show working time：如果勾选了，那么在非 working time 的时间段，Graph 的背景是灰色。这个对饼图和 exploded 饼图无效。

- ◎ Show triggers：如果显示的 Item 有相关的 Trigger，会把相应的报警条件显示在 Graph 中，如图 7-2 所示。

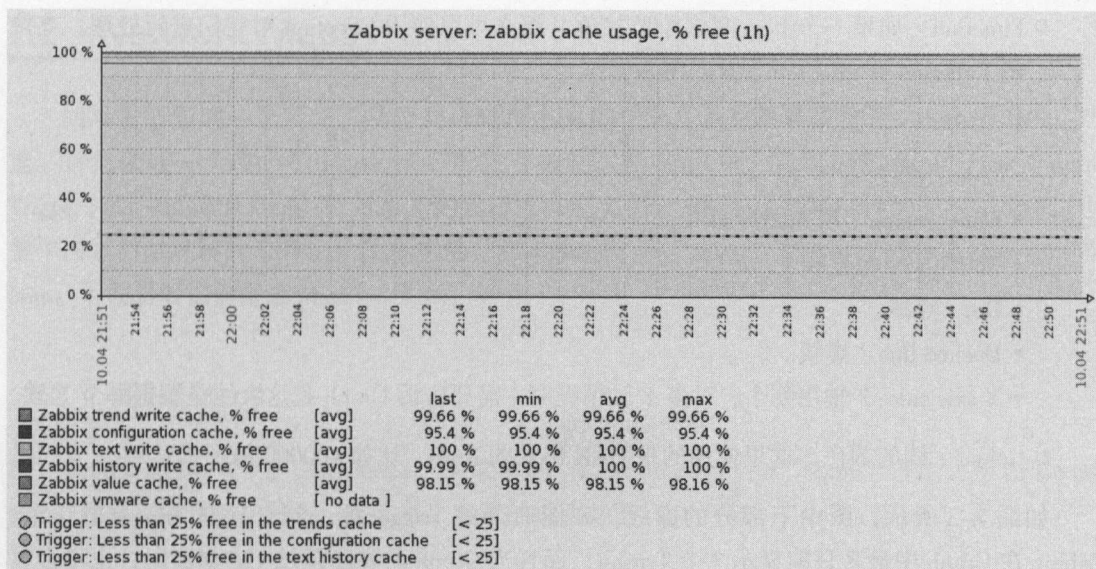


图7-2

- ◎ Percentile line (leftright): 在 Y 轴的左侧 (右侧) 显示百分比, 比如在这里设置了 95%, 那么就会在最大值的 95% 处有一条横线。只对 Normal Graph 生效。
- ◎ Y axis MIN (MAX) value: 这个选项是为了控制 Y 轴的最小值 (最大值) 的, 有以下三个选项。
 - Calculated: 根据一定规则计算出 Y 轴的最小值 (最大值)。
 - Fixed: 固定的 Y 轴最小值 (最大值), 由用户指定。
 - Item: Item 的最近一次监控值会作为最小值 (最大值)。
- ◎ 3D 视角: 只对饼图和 exploded 饼图有效。
- ◎ Items: 设置 Graph 中要显示的 Item。

Graph 的目的是显示 Item 的数据, 那么配置 Graph 具体显示哪些 Item 是关键, 单击 Items 中的 “Add” 按钮, 弹出的窗口有以下属性需要设置。

- ◎ Sort order (0~100): Items 排列的优先级, 0 为最高。优先级越高的, 会显示在越上面的图层, 即优先级高的会遮盖优先级低的。我们也可以使用 Item 前的箭头来调整优先级。
- ◎ Name: Item 的名字。

◎ Function : 如果一个 Item 有多个监控值, 那么可以使用 Function 对它们进行处理。支持的 Function 有 all、min、avg、max。

◎ Draw Style : 绘制数据的形式, 支持的有以下几种。

- Line : 直线。
- Filled region : 填充图形。
- Bold line : 粗直线。
- Dot : 点线。
- Dashed line : 虚线。
- Y axis side: Y 轴用哪个, 对多 Y 轴情况时, 需要告诉 Graph 我这个线是根据哪个 Y 轴。

◎ Colour : 线的颜色, 这里使用的是 HEX 格式的 RGB, 比如 #000000 的形式。

如图 7-3 所示, 图中下部分的虚线, 和图底部的 Trigger 那一行, 就是 Trigger 的信息。Zabbix 在 Graph 中最多只能显示 3 个 Trigger, 而且当 Graph 的高度超过 120 像素时, 也不会显示 Trigger 的信息。

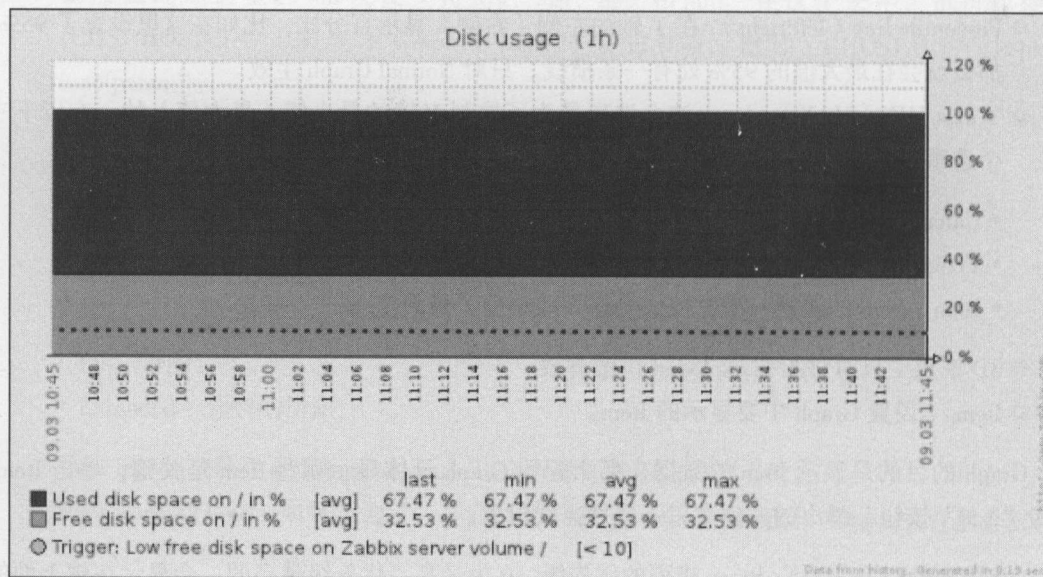


图 7-3

7.2 Network Maps

如果需要管理一个网络，从全局的角度去看整个基础设施，可以使用 Zabbix 的 Maps 功能。首先创建一个空白的 Map，然后配置完善它。在 Maps 中，可以显示 Host、Host Group、Trigger、图片或其他 Map。对 Map 上的元素可以定义一个图标、一些说明性的文字，也可以设置单击它们时跳转的 URL。从菜单栏的“Monitoring”→“Maps”进入可以查看 Zabbix 已有的 Map。下面看看如何配置 Maps。

7.2.1 新建 Maps

首先要新建一个空白的 Maps，从菜单的“Configuration”→“Maps”进入，单击 Create Map。在弹出的窗口中输入一些参数。参数的说明如下。

- Name : 唯一的 Map 名称。
- Width : Map 的宽度，单位是像素。
- Height : Map 的高度，单位是像素。
- Background image : Map 的背景图片，有以下两个选择。
 - No image : 没有背景图片，会使用白色做背景。
 - Image : 选择一张图片作为 Map 的背景。Zabbix 不会对图片进行拉伸。用户可以使用一张地图的图片，使 Map 效果更好。
- Automatic icon mapping : 可以将一个图标和某个 Host 的 inventory 关联起来，从而在 Map 中显示。
- Icon highlighting : 如果勾选了这个选项，那么当 Map 上显示的元素有 Trigger 被触发的元素时，会在这个元素的周围，显示和 Trigger 重要等级颜色相同的圆圈。如果这个元素的状态是“disabled”或者是“in maintenance”，那么会在元素边上显示一个方框。
- Mark elements on trigger status change : 当元素的 Trigger 发生变化时，会在元素的周围有朝内的箭头的边框，显示 30 分钟。
- Expand single problem : 地图上的元素（Host、Host group 或者其他 Map）发生了问题，如果勾选了此，那么出问题的 Trigger name 会显示在元素上。

- ◎ 高级图标：对于不同的元素类型，使用不同的标签。
- ◎ 图标标签类型：不同的图标，可以使用不同类型的标签。具体如下。
 - Label：图标的标签。
 - IP address：IP 地址。
 - Element name：元素的名称，比如 Host name。
 - Status only：只显示 status——OK 或者 PROBLEM。
 - Nothing：不显示任何标签。
- ◎ 图标标签位置：标签相对于图标的位置，可以在图标的底部、左边、右边和顶部。
- ◎ 元素上显示的问题：在元素上会显示一个计数器，显示的是这个元素最近出问题的个数，对于问题的种类，可以做以下几种选择。
 - 全部：所有问题都会使得计数加 1。
 - Separated：没有被 ack 的问题会单独显示。
 - Unacknowledged only：计数器只会显示没有 ack 的问题的计数。
 - 最低 Trigger 严重等级：低于这里设置的等级的 Trigger 不会在 Map 中显示。选择了 Warning 时，Information 等级的 Trigger 是不会有在 Map 显示的。
- ◎ URLs：单击元素时候，可以跳转到一个网页，这里可以定义这个网页的 URL。非常方便的是，在这个配置，可以使用一些宏来定义 URL，支持的宏有 {MAP.ID}、{HOSTGROUP.ID}、{HOST.ID} 和 {TRIGGER.ID}。

7.2.2 创建元素

下面讲解如何添加元素到 Map。

在 Map 的顶部有一条工具栏，单击 Icon 边上的“+”，就会出现一个元素，可以把它拖拽到任何位置。当 Grid 设置为“On”的时候，对于元素的拖拽会对齐于网格线。对于网格线的显示，可以在 Grid 后的下拉框中设置。

单击元素后，会弹出一个窗口，如图 7-4 所示。对这个刚刚新建的元素可以进行一些设置，下面对每一个属性进行说明。

edit map element

Type: Host

Label: New element

Label location: Default

Host: type here to search

Icons

Automatic icon selection ☒

Default: Server (96) Problem: Default

Maintenance: Default Disabled: Default

Coordinates: X: 0 Y: 0

URLs

Name: URL: Remove

Add

Apply Remove Close

图 7-4

- Type : 元素的类型。包括 Host、Map、Trigger、Host group 和 Image。
- Label : 图标的标签, 可以是任何字符串, 可以分多行, 也可以使用宏。
- Label location : 可以选择默认, 就是使用 Map 级别的配置。也可以选择底部、左边、右边和顶部。
- Host : 如果“Type”选择“Host”, 会出现这个输入框, 在这里输入 Host 的名字, 支持自动补全。
- Map : 如果“Type”选择“Map”, 这里选择需要的 Map。
- Trigger : 如果“Type”选择“Trigger”, 这里选择需要显示的 Trigger。
- Host group : 如果“Type”选择“Host group”, 会出现这个输入框, 在这里输入 Host group 的名字, 支持自动补全。
- 默认图标 : 默认显示的图标, 即不做任何配置时显示的图标。
- 自动选择图标 (Automatic icon selection) : 勾选这一项后, Zabbix 会根据服务器的种类、状态自动选择图标来显示。除非有很特殊的需求, 一般建议勾选这个选项。
- 图标 (icon) : 选择不同状态下的元素的图标, 支持的状态有“default”、“problem”、“maintenance”和“disabled”。
- Coordinate X : X 轴坐标。
- Coordinate Y : Y 轴坐标。
- URLs : 设定单击这个元素时跳转的 URL 地址, 可以使用宏。支持的宏有: {MAP.ID}、{HOSTGROUP.ID}、{HOST.ID}、{TRIGGER.ID}。

需要注意的是,在 Map 上增加元素的时候,是不会自动保存的,如果在保存前就离开了页面,那么做的更改就会全部丢失,因为最好能够经常保存 Map。

7.2.3 选择元素

选择一个元素,用鼠标单击就行了,如果要选择多个元素,跟 Windows 中选择多个文件是一样的,按住【ctrl】或者【shift】键,也可以用鼠标拖动一个范围来选中范围内的元素。选中了一个或多个元素后,会弹出一个窗口,可以在里面修改选中的元素的属性。

7.2.4 关联元素

已经在地图上放了一些元素了,可以把它们关联起来。首先选中要关联的两个元素,然后选择 Map 顶部工具栏里 Link 边上的“+”号。

创建了关联后,单击有关联的元素,会弹出配置窗口,下面多了“编辑元素关联”的板块,可以对这个关联关系进行如下配置。

- 标签:在关联这个线上显示的字符串,支持一些宏。
- 连接到:这个关联关系的另一方。
- Type (OK):选择 OK 状态下表示关联关系的线的类型,可以选择“单线”、“粗线”、“点线”或“虚线”。
- Colour (OK):选择 OK 状态下线的颜色。
- Link indicators:对于不是 OK 状态下显示的线的类型和颜色的定义。

7.2.5 关联指示器

“关联”指的是 Map 中两个元素之间建立了关系,我们可以将一个 Trigger 和这个关联关系绑定在一起,当 Trigger 状态发生改变(比如变成了 PROBLEM 状态)时,在 Map 上表示关联关系的线的颜色也会随之改变。如果默认显示的是普通的绿线,当 Trigger 变成 PROBLEM 状态后,线会变成红色的粗线。这样一旦发生问题,就可以从 Map 中很快看到,都不用等邮件报警。

下面看看对于这个需求要如何设置:

- (1) 选择一个 Map 上的元素。

(2) 在 Edit element links 的部分选择 Edit。

(3) 选择 Link indicators 中的 Add, 然后选择一些 Trigger。

现在在 Link indicators 列表中就可以看到刚刚添加的 Trigger 了。在这个界面, 可以对各个 Trigger 的不同状态设置相应的线的颜色和类型。

需要注意的是, 如果一个关联上有多个 Trigger, 那么当多个 Trigger 都变成 PROBLEM 状态的时候, Trigger 严重等级最高的那个, 对其设置的关联的线的颜色和类型会优先显示。如果 Trigger 严重等级相同, 那么越小 ID 的 Trigger 所对应的设置越会优先生效。

7.3 Screens

Zabbix Screen 是用来将一些 Graph 组织到一起展示的工具。有时想一眼就能看到一台服务器的多个数据, 或者是多台服务器的同一个数据, 对于这种需求, 如果只依靠之前了解到的 Graph 是很困难的, 可能需要打开好几个窗口来回地看。有了 Screen 就能摆脱这种困境了。下面先看一个 Screen 的示意图, 如图 7-5 所示。

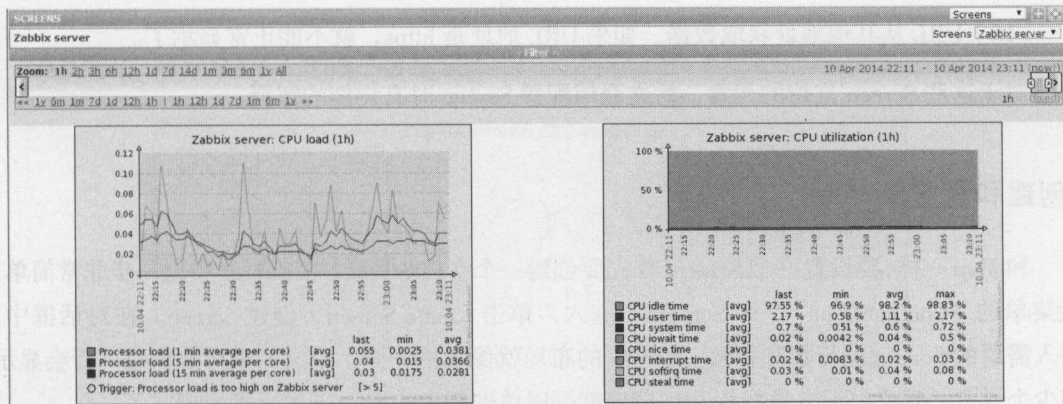


图7-5

可以把一个 Screen 理解为一个 Table, 可以设置 Screen 有多少行、有多少列, 以及每一个“单元格”内显示的内容, Screen 支持如下各项。

(1) simple graph: 最基本的 Graph, 在 Latest Data 中单击 Graph 后即可看到。

(2) 用户自定义的 Graph。

- (3) Maps。
- (4) 其他 Screen 大屏幕。
- (5) 文本信息。
- (6) Server information (overview): 总览的 Zabbix Server 信息。
- (7) Hosts information (overview): 总览的 Host 信息。
- (8) Trigger information (overview): 总览的 Trigger 信息。
- (9) Host/Host group issues (status of triggers): 某个 Host 或者 Host group Trigger 的状态。
- (10) System status: 系统状态。
- (11) Data overview: 数据状况。
- (12) Clock: 时间。
- (13) History of events: 事件的历史。
- (14) History of action: 报警动作的历史。
- (15) URL 从其他地址获取数据, 如果 URL 地址是 https, 就不能正常显示了。

对于这些 Screen 支持的元素, 后文介绍配置 Screen 时有详细的解释。

创建和配置Screen

和 Map 一样, 要设置一个 Screen, 首先要创建一个空白 Screen。在 Zabbix 中这一步非常简单。从菜单的 “Configuration” → “Screens” 进入, 单击 Create Screen, 创建 Screen。在对话框中, 输入需要的 Screen 的行数和列数。Screen 的布局就像表格一样, 定义的是行数和每一行会显示多少个元素。定义好后, 选择保存。接着就可以单击新建的 Screen 进入编辑模式了。

因为这个 Screen 是新建的, 所以每一个 “单元格” 内都是空的, 只有一个 “Change” 链接, 单击后, 会弹出一个窗口, 定义这个位置的单元格应该显示的内容, Horizontal align 和 Vertical align 都是设置对齐的方式, Column span 和 Row span 与 HTML 中的 span 一样, 表示合并 “单元格”。比如有一张图特别宽, 那么可能需要将两个单元格合并显示在其中。Resource 是定义要显示什么的参数, Resource 可选的有以下几种。

◎ Clock : 显示当前时间, 是一个拟物的时钟, 如图 7-6 所示。

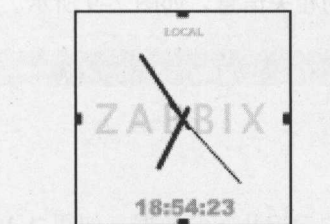


图 7-6

◎ Data overview : 显示一组 Host group 的某些 Application 的 Item 的值, 如图 7-7 所示。

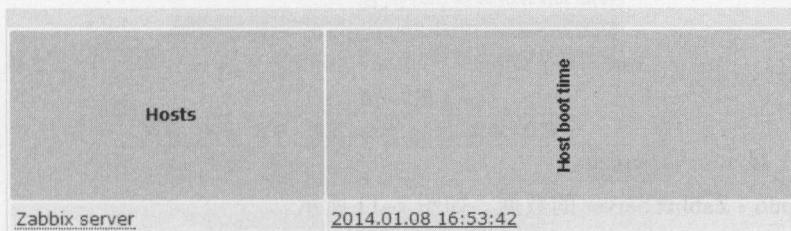


图 7-7

◎ Graph : 某个 Graph, 如图 7-8 所示。

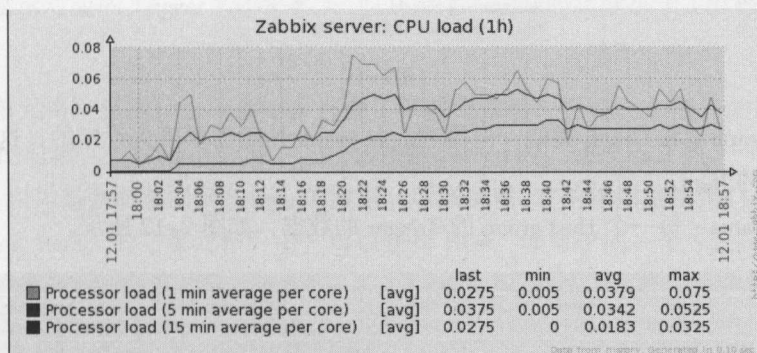


图 7-8

- ◎ History of action : 最近 Zabbix 执行的 action 历史。
- ◎ History of event : 最近 Zabbix 发生的事件。
- ◎ Host group issues : 一个 host group 中的所有 Triggers 的状态。

- ◎ Host issues : 一个 Host 的所有 Triggers 的状态。
- ◎ Host info : 高层次的 Host 的相关信息，如图 7-9 所示。

Hosts info Group "Zabbix servers"			
1 Available	0 Not available	0 Unknown	1 Total

图7-9

- ◎ Map : 显示一个 Map。
- ◎ Plain text : 显示一个 Item 的最近几次的的数据。如图 7-10 所示。

Timestamp	Zabbix server: Agent ping
12 Jan 2014 19:03:07	Up (1)
12 Jan 2014 19:02:07	Up (1)
12 Jan 2014 19:01:07	Up (1)
12 Jan 2014 19:00:07	Up (1)
12 Jan 2014 18:59:07	Up (1)

图7-10

- ◎ Screen : 显示一个 Screen。
- ◎ Server info : Zabbix Server 的数据，如图 7-11 所示。

Zabbix server info
Updated: Sun, 12 Jan 2014 19:04:21 +0800
Users (online): 2(1)
Logged in as Admin
Zabbix server is running
Hosts (m/n/t): 39(1/0/38)
Items (m/d/n): 76(69/0/7)
Triggers (e/d)[p/o]: 46(46/0)[0/46]

图7-11

- ◎ Simple graph : 和 Graph 类似，但是 Simple graph 是不需要事先设定的，这里可以选择某个 Host 的 Item。
- ◎ System status : 每一个 Host group 的 Trigger 的数据，如图 7-12 所示。

Status of Zabbix						
Host group	Disaster	High	Average	Warning	Information	Not classified
Zabbix servers	0	0	0	0	0	0
Updated: 19:06:23						

图7-12

- ◎ Trigger info : 某些 Host group 的 Trigger 数据，如图 7-13 所示。

Triggers info Group "Zabbix servers"						
46 Ok	0 Not classified	0 Information	0 Warning	0 Average	0 High	0 Disaster

图7-13

◎ Triggers overview : 某些 Host group 的某些 Application 的 Trigger 数据, 如图 7-14 所示。

Host	Host information was changed on {HOST.NAME}	Hostname was changed on {HOST.NAME}	{HOST.NAME} has just been restarted
Zabbix server			

图 7-14

◎ URL : 显示一个网页, 注意, 要加上 “http://”, 如图 7-15 所示。

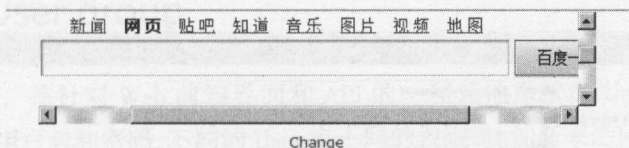


图 7-15

在“表格”的四个边, 有“+”和“-”, 表示在边上加一行(列)和减一行(列)。选择 Graph 时, 在 Graph 上会显示对应的 Trigger (如果有), 但是如果 Graph 的高度小于 120 像素, 就不会显示 Trigger。

在配置“单元格”显示什么元素时, 选择了某些元素后, 会有一个额外的选项“Dynamic item”, 勾选它的时候, 什么都不会发生。我在接触 Screen 时, 还没有文档, 一直不明白这个是干什么用的, 以为是会动态刷新 Screen 的数据。后来发现, 它的作用是使我们浏览 Screen 时, 在勾选“Dynamic item”的“单元格”中, 可以选择 Host。即对于同一个 Screen, 不同的人可以选择不同的数据来看, 也免去重复建立 Screen 的痛苦。当 Resource 定义为“Graph”、“Simple Graph”、“Plain Graph”时生效。

7.4 Slide shows

Zabbix 中的幻灯片, 指的是多个 Screen 以设定的时间间隔进行切换。有的时候需要在几个 Screen 中来回切换, 比如对于每一个 Nginx 域都做了一个 Screen, 那么运维人员需要在几个主

要域的 Screen 间切换，这就是 Slide show 幻灯片的用处。

首先，到“Configuration”→“Slide shows”中，单击“Create slide”按钮，进行如下一些配置。

- Name : Slide 的名字。
- Default delay (in seconds) : 切换幻灯片的时间间隔。
- Slides : 需要在 Slides 中显示的 screens。
- Screen : Screen 的名字。
- Delay : 可以单独对某个幻灯片设置显示的时间，如果设置为 0，会使用 Default delay。
- Action : 单击可以移除幻灯片。

如图 7-16 中的配置，幻灯片的显示顺序就是：Zabbix server- 显示 30 秒 -New host- 显示 15 秒 -Zabbix server- 显示 30 秒，接下来就一直循环。

Slides	Screen	Delay	Action
1	Zabbix server	default	Remove
2	New host	15	Remove

图 7-16

设置好以后，可以在“Monitori”→“Slide shows”中看到所有的幻灯片。在看幻灯片的地方，有一个下拉框，叫做 Refresh time multiplier，它能够加快或者减慢幻灯片播放的速度。

第 8 章

Users和Macros

8.1 User和User group

User 这个概念，是针对 Web 前端界面和 API 的。和其他系统的 User 一样，Zabbix 中的 User 需要有自己的用户名和密码，不同的 User 有不同的权限，比如新来的同事在熟悉 Zabbix 时，只能看看数据而不能真正去修改配置。

而对于 User group，它和 User 的关系，类似于 Host group 和 Host 的关系，但 User group 和 User 的关系更紧密。这是因为我们可以对 User group 定义一些配置，这些配置对 User group 下的所有 User 生效。

本章主要介绍如何配置 User、User group 以及权限相关。

8.1.1 配置User

配置 User 主要有三块内容：User 的基本信息、Media 联系方式和 Permission 权限。

从菜单栏进入“Administration”→“Users”，在页面的右侧上方下拉框选择“Users”，单击“Create user”（如果是修改现有的 User，那么单击现有 User 的用户名即可）。

首先看到的是配置 User 基本信息的标签，下面看看每个参数的作用。

- Alias：唯一的用户名，登录时使用。
- Name：用户的名字，可选属性，它会在确认事件的时候和通知信息的时候显示。

- ◎ Surname : 和 Name 相同。
- ◎ Password : 密码。
- ◎ Groups : 设置该用户属于的 User group。
- ◎ Theme : 前端的主题，主要是不同的配色方案。
- ◎ Auto-login : 登录过一次后，30 天内不需要再输入密码登录。
- ◎ Auto-logout (min 90 seconds) : 90 秒后自动退出登录。
- ◎ Refresh (in seconds) : 打开 Graphs, Screens, Plain text 数据时，Zabbix 会自动刷新页面。
如果设置为 0，则禁止该功能。
- ◎ Rows per page : 每一页显示的数据的行数。
- ◎ URL (after login) : 登录后直接跳转到该 URL 的页面。

Zabbix 安装完成之后有两个用户可用，一个是“Admin”，一个是“guest”。“guest”用户不需要登录就可以进入 Zabbix，出于安全考虑，最好禁用。虽然 guest 用户只能看监控数据，但是公司的监控数据其实也是非常敏感的，有时候还能推算出一些财务数据。比如可以通过服务器的网卡流量估计网络带宽、IDC 架构、服务器数量等。

User Media 标签用来设置用户的 media，这里不做展开。

Permissions 标签可以看到这个用户目前对于 Host 和 Host group 的权限情况。下面对“User type”的下拉框的三个选项做一下说明。

(1) Zabbix User : 该用户可以访问菜单栏中的 Monitoring 板块。用户对于所有资源默认都没有访问权限，所有可以访问的 Host 或者 Host group 都需要指定。

(2) Zabbix Admin : 该用户可以访问菜单栏中的 Monitoring 板块和 Configuration 板块。用户对于所有资源默认都没有访问权限和修改权限，所有可以访问和修改的 Host 或者 Host group 都需要指定。

(3) Zabbix Super Admin : 最大权限组。该用户可以访问 Zabbix 的所有板块。用户对于所有资源默认有修改权限。如果想禁止对某个资源的访问，需要显式地注明。

简而言之，Zabbix User 是只能看不能改，并且要设置它能看什么。Zabbix Admin 是又能看又能改，但是要设置它能看什么和修改什么。Zabbix Super Admin 对于所有的东西都能看都能改，可以设置它不能看什么。

需要注意的是，在权限这方面，都是在 Group 层级设置的。即只能在 User group 上定义针对 Host group 的权限控制，不能定义某个 User 能（或不能）查看某个 Host 的数据。

一个 User 可以属于任意多个 User group, 那么在这种情况下, 如何决定一个 User 对于一个 Host group 的权限呢? 比如, HostX 属于 Hostgroup1, 而 User1 属于 User group A 和 User group B, 这里会有各种情况, 我们看看 Zabbix 是如何处理的。

(1) 如果 GroupA 对 HostX 有 Read 权限, GroupB 对 HostX 有 Read-Write 权限, 那么 UserA 对 HostX 有 Read-Write 权限。

(2) 如果 HostX 还在 Hostgroup2 中, 而 GroupA 和 GroupB 对 Hostgroup2 是禁止的, 那么 UserA 对于 HostX 也是禁止的。

(3) 如果 GroupA 对于 HostX 没有任何定义, 而 GroupB 对 HostX 有 Read-Write 权限, 那么 UserA 对于 HostX 有 Read-Write 权限。

(4) 如果 GroupA 对于 HostX 是禁止的, 而 GroupB 对于 HostX 有 Read-Write 权限, 那么 UserA 对于 HostX 是禁止的。

因此, 对于这种 User 属于多个 User group 而需要不同权限的问题时, 有以下两个原则。

(1) 优先级大小: denied > Read-Write > Read。

(2) denied 是排他性的, 一旦有一个 User group 对于 Host 是 denied 的, 那么, 无论如何, User 对其也是 denied 的。

8.1.2 User group

前面把 User group 和 Host group 放在一起介绍, 一是因为它们都是一个分组功能的实现: User group 是一组 User, Host group 是一组 Host; 二是因为权限的设置是在 User group 和 Host group 之间的事情。权限的问题, 前面已经介绍过, 下面主要是介绍 User group 的配置。

在 User group 的配置页面中, 有两个标签, 首先看设置基本信息的 User group 标签的配置。

- Group name: 唯一的 User group 的名称。
- Users: 属于这个 User group 的 User。
- Frontend access: 该 User group 的 User 登录 Zabbix 方式的设置。
 - System default: 使用默认的认证方式。
 - Internal: 使用 Zabbix 的认证方式。如果在 HTTP authentication 中设置了其他方式, 那么选择这个将会无效。

- Disable：禁止该用户访问 Zabbix 前端（只能通过 API 访问）。
- ◎ User status：该 User group 中的 User 是否可用。
 - Enabled：可以使用。
 - Disabled：不可以使用。
- ◎ Debug mode：是否打开调试模式。

在 Permission 标签中，有以下设置。

(1) Composing permissions：有三列，分别为 Read-Write 读写权限、Read only 只读权限和 Deny 禁止访问，可以在任意一列添加 Host group。

(2) Calculated permissions：Composing permissions 中在不同条件下添加了一些 Host group，根据这些设置，Zabbix 会计算出这个 User group 对哪些 Host 和 Host group 有何种权限。

8.2 Macros

我最早接触“宏”这个概念，是在 Word 里。微软的 Office 官网上对“宏”是这样介绍的：“宏是通过一次单击就可以应用的命令集。它们几乎可以自动完成您在程序中执行的任何操作，甚至还可以执行您认为不可能的任务。”在 Word 里，可以使用 VBA 语言来编写复杂的宏；在 C 语言里，也有“宏”这个概念，它的作用简单来说和 Word 里的宏差不多，就是事先定义好一些东西，在其他地方使用时，就不需要重复劳动了。

8.2.1 自带宏

Zabbix 中的宏与 Word 中的宏的作用类似，不同的是，Zabbix 中一部分宏可以事先定义，而另一部分宏是和具体的场景有关的。

事先定义的宏，是用户在全局定义的，操作路径是“Administration”→“General”→“Macros”。而和场景有关的宏才是使用最多的。比如，在报警内容中，就使用了大量的宏：

```
Trigger: {TRIGGER.NAME}
Trigger status: {TRIGGER.STATUS}
Trigger severity: {TRIGGER.SEVERITY}
```

```
Trigger URL: {TRIGGER.URL}
```

```
Item values:
```

```
(1) {ITEM.NAME1} ({HOST.NAME1}:{ITEM.KEY1}) : {ITEM.VALUE1}
```

```
(2) {ITEM.NAME2} ({HOST.NAME2}:{ITEM.KEY2}) : {ITEM.VALUE2}
```

```
(3) {ITEM.NAME3} ({HOST.NAME3}:{ITEM.KEY3}) : {ITEM.VALUE3}
```

```
Original event ID: {EVENT.ID}
```

大家应该可以发现，Zabbix 中的宏，是在尖括号中的，中间的字符串以点分隔，表示层级关系。以 {TRIGGER.NAME} 为例，这个宏表示的意思就是 TRIGGER 中的 NAME 属性。

Zabbix 的宏有什么用呢？如果要写一个监控本机 8080 端口是否存活的监控，但是 8080 端口并不是起在 0.0.0.0 或者是 127.0.0.1 上的，而是绑定在自己内网 IP 上，比如 10.1.1.1:8080，所以在写 Item 时，需要指定机器的 IP 地址。对于这种监控，肯定是放在 Template 上定义的，否则每台机器定义一遍，会很麻烦。最好的方法就是在 Template 上定义一个 Item，其中 IP 是个变量。key 形如 “net.tcp.port[{HOST.IP1},8080]”。这样，在每一台服务器上，都会根据自己的 IP 来监控。

并不是每个宏都可以在所有的地方使用的，不同的宏有不同的适用场景，Zabbix 提供了一份表格，上面写明了哪些宏在哪些地方可以适用。笔者选取了一些常用的放在本书的附录，完整的可以在 Zabbix 官方文档附录中找到。

8.2.2 用户自定义宏

除了 Zabbix 自带的宏，还可以进行自定义。自定义宏分不同的等级，优先级的排列顺序如下。

(1) Host 级别的宏。

(2) Template 级别的宏。

(3) 全局宏。

在使用时，格式为 {SMACRO}，其中宏的名字只能使用 A ~ Z 和 0 ~ 9 来表示。引用宏以后，Zabbix 会从 Host 级别开始找，然后是 Template 级别，最后是全局宏。如果都没有找到匹配的，就会直接显示形如 “{SMACRO}” 的字符串。

如何定义想要的宏呢？全局宏可以在 “Administration” → “General” → “Macros” 中定义。而对于 Host 或者 Template 级别的宏，则是在 Host 和 Template 配置的 Macros 标签来定义。

的。其实我们更倾向于在 Template 这个级别定义宏，因为在管理大量服务器的时候，很少会在 Host 上建立一些特定的 Items 或者 Triggers，这样会非常不利于管理，应该把配置全部放在 Template 这个级别上。宏也是一样，定义 Host 级别的宏维护起来非常困难，总不见得以后加一个 Host 就要在上面配置一个宏吧？配置在 Template 上还有个好处，就是在导出 Template 的时候，可以连带将定义在 Template 上的宏一起导出。

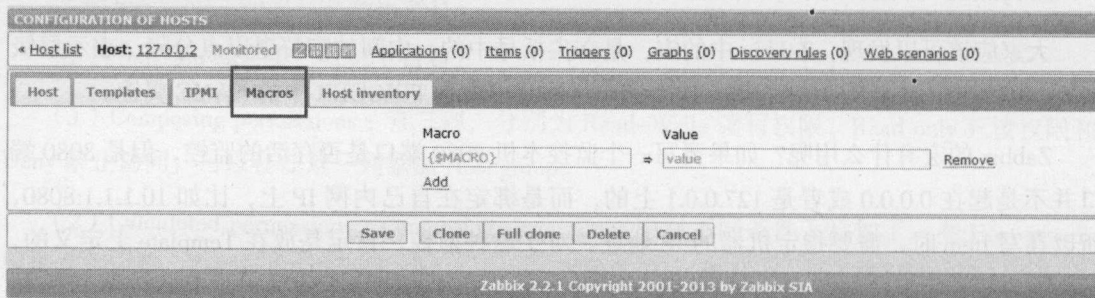


图8-1

MySQL 是默认运行在 3306 端口的，由于特殊需求，还有运行在 3307 上的，那就需要两个 Template，分别针对 3306 和 3307。假设 Item 叫做 mysql[<mysql_port>]，那就要在两个模板上定义几乎相同的 key:mysql[3306] 和 mysql[3307]，一个 Item 还好，如果有几十个呢？就会非常麻烦。这时，可以使用 Template 级别的宏，在 3306 端口的模板上定义一个 {MYSQL.PORT}=3306，同样，在 3307 端口的模板上定义一个 {MYSQL.PORT}=3307，而 Item 的 key 就变成相同的 mysql[{MYSQL.PORT}] 了。这样，这两个模板的所有 Items 和 Triggers 都是一样的，不同的只是这两个宏。是不是简单很多呢？

在什么地方可以用自定义宏呢？Zabbix 给出的答案是下面这三个。

- (1) Item 的 key 和名称。
- (2) Trigger 的名称和表达式。
- (3) 其他特定的地方。

其实使用宏最主要的地方就是 (1) 和 (2)。(3) 的内容会在 8.2.3 小节列出。

在 Trigger 中使用自定义宏有两个地方要注意：

- (1) 自定义宏可以作为 expression 中函数的参数，比如定义了 {MAX_TIMES}=#5，那么就能使用 min ({MAX_TIMES})。

(2) 自定义宏可以在 expression 中使用, 当且仅当它表示的是一个常量时。不能使用自定义宏去表示一个 Hostname、Item 的 key、函数和 operator。比如定义了 {FUNC}=min, 然后想像 {FUNC}(#5) 这样使用, 是不允许的。

8.2.3 自定义宏的适用范围

自定义宏除了上文提到的 Items 和 Triggers, 还可以在下面这些地方使用。

Hosts	<ul style="list-style-type: none"> • Interface IP/DNS • Interface port
Passive proxy	<ul style="list-style-type: none"> • Interface port
Items	<ul style="list-style-type: none"> • SNMPv3 security name • SNMPv3 auth pass • SNMPv3 priv pass • SNMPv1/v2 community • SNMP OID • SSH username • SSH public key • SSH private key • SSH password • Telnet username • Telnet password • Calculated item formula • Trapper item “Allowed hosts” field (since Zabbix 2.2)
Discovery	<ul style="list-style-type: none"> • SNMPv3 security name • SNMPv3 auth pass • SNMPv3 priv pass • SNMPv1/v2 community • SNMP OID

第 9 章

IT services服务监控与 Web monitoring网络监控

9.1 Services服务监控

在上面几个章节，我们讲解了 Zabbix 如何对 Host 进行监控，这是面对一线工程师和一线经理的，但对于老板来说，他可能并不关心某台服务器是否正常，或者说某个网络设备流量是否跑满，他关心的是公司的服务是否正常。本章，首先从运维角度，介绍服务层本身和监控的必要性，再介绍 Zabbix 中如何设置服务层监控。

运维架构由下而上来看，第一层是设备层，即服务器、网络交换机等；第二层是应用层，即各种应用，如 MySQL、Resin 等开源工具；第三层是服务层，这在各个公司是不同的，比如 PPTV 中的点播服务、评论服务等。Zabbix 除了支持设备层和应用层外，还支持服务层的监控。设备层，即各类基础监控，比如 CPU 负载、磁盘空间等，对于应用服务，除了 Zabbix 自带的一些监控外，还可以自定义脚本。而对于服务层，情况就稍微有点复杂了，因为服务并不是一个简单的监控，它是一个树状结构，比如一个服务依赖于多个机房中的多个机器，或者依赖于一组服务器，就算其中一些服务器出了问题，服务还会是正常的。

通过上面这一段简单的介绍，相信各位已经明白，“服务”的内涵以及监控的困难点。Zabbix 对于 IT service（即服务），提供了树状结构的定义来解决这些问题。下面看一个简单服务的例子。

```

IT Service
|
|-Workstations
| |
| |-Workstation1
| |
| |-Workstation2
|
|-Servers

```

上面的代码中，“IT Service”是一个服务，它依赖着“Workstations”，而“Workstations”又依赖于两台 Workstation：“Workstation1”和“Workstation2”。在 Zabbix 中，这里的“IT Service”、“Workstations”、“Workstation1”、“Workstation2”和“Servers”都被称为节点，每一个节点都有自己的状态。对于节点之间的依赖关系，可以进行定义。比如“Workstations”依赖“Workstation1”和“Workstation2”，那么可以定义当“Workstation1”和“Workstation2”有一台正常时，节点“Workstations”就是正常的；也可以定义“Workstation1”和“Workstation2”都正常时，“Workstations”节点才能是正常的。

9.2 服务配置

下面会将上一节的例子具体化，然后使用 Zabbix 来实现。

代码如下：

```

Passport Service
|
|-MySQL
| |
| |- Slave1
| |
| |- Slave2
|
|-Servers
| |

```

```

| |-Server1
| |-Server2

```

整个服务叫做“Passport Service”，它依赖“MySQL”和“Servers”，“MySQL”依赖于两个 Slave 实例：“Slave1”和“Slave2”，这两台 Slave 只要有一台是正常的，那么“MySQL”就是正常的。而对于“Servers”来说，只有当“Server1”和“Server2”都正常的情况下，它才是正常的。

接下来开始逐步去完成这个 IT service 的监控。

首先从菜单栏“Configuration”→IT services“进入。如果是第一次进入，会发现已经有了一个叫做“root”的 Service 存在。这个和 Zabbix 本身的设置有关，所有的服务都是以“root”作为父节点的。

首先建立“Passport Service”，步骤如下。

（1）单击“root”，选择“Add service”，就能配置要增加的 service 了。目前我们只在“Name”上输入“Passport Service”，单击“Save”。然后可以看到，在“root”下多了一个子节点“Passport Service”。如图 9-1 所示。

（2）创建“MySQL”和“Slave1”、“Slave2”，以及“Servers”、“Server1”、“Server2”。大家要记住单击的节点，就是新建的节点的父节点（当然，也可以在新建时选择）。按照（1）中的方法新建好整个树状结构。如图 9-1 所示。

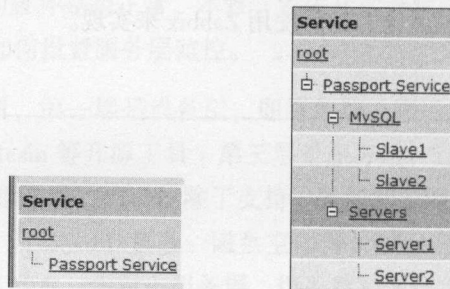


图9-1

下面看看对于一个 service，Zabbix 可以配置的参数，如图 9-2 所示。

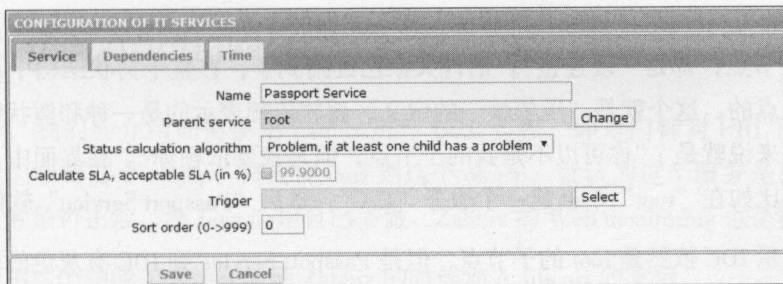


图9-2

- Name : 服务的名字。
- Parent service : 父节点。
- Status calculation algorithm : 这是个比较重要的参数, 表示的是这个节点的状态, 是如何依赖于子节点的。
 - Problem, if as least one child has a problem : 只要有一个子节点异常, 该节点就异常。
 - Problem, if all children have problems : 当且仅当所有子节点都异常, 该节点才异常。
 - Do not calculate : 不计算节点的状态。
- Calculate SLA, acceptable SLA (in %) : 是否计算 SLA 的百分比, 在 Report 中使用。
- Trigger : 该节点的异常状态所依赖的 Trigger。在最底层的服务, 一定要依赖 Trigger, 否则节点状态会显示不对。
- Sort order : 显示的顺序, 数字小的优先。

基本配置介绍完了, 接下来要继续完善服务。前面已经把树状结构在 Zabbix 建好了, 现在只要把出问题的规则配置一下即可。

“Slave1” 和 “Slave2” 只要有一台正常, “MySQL” 就是正常的, 那么在 “MySQL” 中配置的算法就是 “Problem, if all children have problems”; 而对于 “Servers”, 只要子节点有一个出问题, 它就有问题了, 所以选择的算法是 “Problem, if at least one child has a problem”。

对于最底层的节点 (比如 “Slave1”), 如果要配置它什么情况下异常, 什么情况下正常, 那么就配置它的 “Trigger” 参数, 将它和一个 Trigger 绑定在一起即可。

服务已经完全配置好了, 整个过程比较简单。接下来一起学习下更复杂的配置。

在 Dependencies 标签, 可以看到该节点的所有子节点, 另外, 在 “Soft” 的选择框下, 有 “Soft Dependency” 和 “Hard Dependency” 两项, 即 “软依赖” 和 “硬依赖”, 这名字是不是让人觉

得和 Linux 中的“软连接”“硬连接”很像？其实它们的含义确实类似。Linux 中，默认我们在之前添加的子节点，都是“硬连接”。估计大家已经猜到了，在整个树状结构中，节点是硬依赖于它的子节点的，这个就是“硬依赖”的定义。而软依赖表示的是一种和树状结构无关的依赖关系。简单来说就是：“你可以不是我的子节点，但是我要依赖你”。在界面中，软依赖是显示灰色的线。比如在“root”下新建一个服务“IDC”，然后“Passport Service”软依赖于它。

图 9-3 所示 IDC 依然是 root 的子节点，但是 Passport Service 到 IDC 有灰色的软依赖。

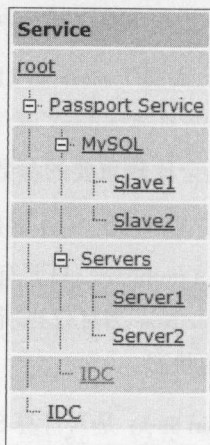


图9-3

对于 Time 标签，这个就比较简单了，它的用处是设置该节点的工作时间或者维护时间。对于所有服务，默认是 7×24 小时，这个会影响到计算 SLA 的可用性。

有三种方式可以设置 Time 标签。

(1) Uptime：设置服务何时应该正常。

(2) Downtime：设置维护时间，以后每周这段时间内的异常，不会计算入 SLA 百分比。

(3) One-time downtime：一次性的维护时间。和 Downtime 不同，One-time downtime 设置的可以具体到某一天的维护时间，仅此一次。

另外，当我们要删除一个节点的时候，首先需要删除它的所有子节点。但是对于有软依赖的节点，删除它之前不需要删除其软依赖的目标节点。

9.3 Web monitoring网络监控配置

在这一节，我们会介绍如何使用 Zabbix 进行 URL 监控，即专门针对 URL 的监控。比如 Java 应用，我们需要有个 ok.jsp，只要能 curl 到这个 ok.jsp，就认为这个服务是正常的。而如何判断这个服务是否正常，由 Java 应用自己去做。Zabbix 的 Web monitoring 也是类似的功能。

如果要使用这个功能，必须在编译 Zabbix 的时候加入 libcurl 的支持。

在开始之前，先介绍 Zabbix 的一个概念——“Web scenarios”，即网络场景，就是“用户为了完成某个目标，在网络上进行的一系列操作的过程”。比方说，要登录微博，就是一个“网络场景”，在其中的每一步操作——打开页面、发起登录请求、登录成功或者失败，在 Zabbix 中被统称为“步骤”。定义了一个 Web scenario，我们需要定义其中有什么步骤，以及按照什么顺序执行，然后 Zabbix 会定期按照设置的顺序来执行这些“步骤”。

在 Zabbix 2.2 中，Web scenario 和 Items、Triggers 等一样，是依附在 Hosts/Templates 上的。目前 Zabbix 对所有 Web scenario，会收集下列数据。

(1) 所有“步骤”的平均下载速度。

(2) 失败的“步骤”。

(3) 最后一次出错信息。

对 Web scenario 中的所有步骤，都会收集下列数据。

(1) 平均下载速度。

(2) 相应时间。

(3) HTTP 状态码。

除此之外，Zabbix 还支持获取指定 URL 的 HTML 内容，看其中是否存在某个字符串。甚至还可以模拟登录动作，模拟鼠标单击。Zabbix 同时支持 HTTP 和 HTTPS。在一个 Web scenario 中，还会保存 cookie，做到真实地模拟一次完整的访问。

下面具体讲解如何配置。

随便找一个 Host 或者 Template，在上面新建一个 Web scenario，配置界面如图 9-4 所示。

图9-4

- ◎ Host：这个不能改，是 Web scenario 所属的 Host 或者 Template。
- ◎ Name：Web scenario 的名字，支持宏。
- ◎ Application：和 Item 的 Application 一样，作为一个分类。
- ◎ New Application：新建一个 Application，并且这个 Web scenario 属于它。
- ◎ Authentication：验证。如果 URL 需要身份验证，Zabbix 同样支持三个下拉选项，除了 None 外，另外两个为 Basic authentication 和 NTLM authentication。选择非 None 的选项时，会多出两个对话框，需要输入登录验证时的用户名和密码，在这两个对话框中，是支持宏的。
- ◎ Update interval(in sec)：执行的时间间隔。
- ◎ Retries：Web scenario 中每一个步骤执行的重试次数。由于网络波动（在我国错综复杂的网络情况下尤其常见）造成某一个步骤执行出错（或者超时等）。这里的设置对于 Web scenario 中的每一个步骤都有效，最大是 10，默认是 1。需要注意的是，并不是只要错误的 HTTP 状态码（比如 404）或者期望的字符串没有出现就会触发这个重试的。
- ◎ Agent：这里的 Agent 就是 Nginx 日志中的 Agent，可以设置模拟使用某种浏览器发起请求。选择“(other...)”则可以手动输入 Agent。这是个非常有用的参数，特别是针对那些只对某种浏览器出问题的 URL。
- ◎ HTTP proxy：如果使用 HTTP 代理，可以使用这个。书写的格式是：`http://[username[:password]]@proxy.mycompany.com[:port]`，默认使用 1080 端口。这里粗看是个 HTTP 代理，其实，做复杂了，甚至可以实现类似基调 workbench 的各个 IDC 对某个 URL 的监控。比如，

可以使用广东机房的代理去访问某个 URL, 那么获取到的 QOS 数据就是广东的了。

◎ Variables : 定义 Web scenario 级别的变量, 或者宏。可以在 URL、Post 等参数中使用。使用如下的格式设置。

- 针对宏 : {macro1}=value1
- 针对变量 : {username}=Alexei

这里有一个特殊的地方, 即支持 regex 语法。比如 {hostid}=regex:hostid is ([0-9]+), 表达的意思是在返回的 HTML 中搜索符合“hostid is ([0-9])”的字符串, 并把匹配到的字符串组里的内容复制给 hostid 这个变量, 如果 HTML 里有“hostid is 12345”, 那么变量 hostid 就是 12345。正则表达上至少有一个组。regex 语法, 不只针对变量支持, 对于宏, 也是支持的。

要注意的是, 这些变量, 并没有经过 URL 编码处理。

什么是“URL 编码”呢? 这里简单解释一下, 它就是将一些网页地址保留的特殊字符转换为非特殊字符。“%”就是 URL 编码后的产物。比如“a=b”在 URL 编码后, 就变成“a%3Db”, URL 编码将“=”变为了“%3D”。

◎ Enabled : 是否启用这个 web scenario

讲解了 Web scenario 的基本设置后, 接下来看看关键的“步骤”是怎么设置的。

切换到“Steps”标签, 单击“Add”按钮, 可以看到弹出的窗口如图 9-5 所示。

The image shows a configuration window titled "Step of scenario". It contains the following fields and values:

- Name:** Home
- URL:** http://www.google.com
- Post:** (empty text area)
- Variables:** (empty text area)
- Timeout:** 15
- Required string:** (empty text field)
- Required status codes:** 200

图9-5

其中各项的具体内涵如下。

- Name : 唯一的名字, 可以使用宏。
- URL : 需要检查的 URL, 支持 HTTP 和 HTTPS, GET 参数可以直接写在 URL 中。比如 `http://zabbix.com?time=10`。
- Post : POST 请求中的 post 的变量, 格式为: “name=frank&age=25&home={home}”, {home} 是在 Web scenario 中设置的宏。同样, 这些数据也是没有经过 URL 编码的。
- Variables : 和 web scenario 的 Variables 类似, 不同的是这里设置的变量, 只在这个步骤中有效, 并且只在 GET、POST 参数中使用有效。
- Timeout : URL 的超时时间。这里设置的超时时间, 并不是检查这个 URL 的总时间, 而是分两部分: 一部分是在连接这个 URL 时会使用的超时时间, 另一部分是发起 HTTP 请求得到反馈的超时时间。也就是说, 一个 URL 检查最多会耗费两倍这里设置的超时时间。
- Required string : 这里需要配置一个正则表达式。当获取到这个 URL 的返回内容时, Zabbix 会根据这个正则表达式在返回的 HTML 中寻找。
- Required status codes : 设置期望的 HTTP 状态码, 这里可以写多个, 比如 200,201,210-299 这种书写格式。

注意, 关于“步骤”的任何修改, 只有当它属于的 Web scenario 保存了才可以生效, 如果在“步骤”这里设置了很多, 但是没有在 Web scenario 中保存, 那就白费工夫了。

9.4 监控百度示例

Web scenario 的配置在前面已经介绍完了, 下面一起做一个监控百度的例子。

首先新建一个 Web scenario 和它的 step。step 很简单, 有两个, 只要分别写上 URL 就行了, 一个 URL 是 “`http://www.baidu.com`”, 另一个 URL 是 “`www.vip.com`”。在 “Monitoring” → “Web” 或者 Latest data 中查看到的界面如图 9-6 所示。

Download speed for scenario "web test".	09 Feb 2014 10:39:30	10.77 KBps
Download speed for step "baidu home page" of scenario "web test".	09 Feb 2014 10:39:20	1.51 KBps
Download speed for step "vip home page" of scenario "web test".	09 Feb 2014 10:39:30	20.04 KBps
Failed step of scenario "web test".	09 Feb 2014 10:39:30	0
Response code for step "baidu home page" of scenario "web test".	09 Feb 2014 10:39:20	200
Response code for step "vip home page" of scenario "web test".	09 Feb 2014 10:39:30	200
Response time for step "baidu home page" of scenario "web test".	09 Feb 2014 10:39:20	10s 160.1ms
Response time for step "vip home page" of scenario "web test".	09 Feb 2014 10:39:30	10s 138.7ms

图9-6

这里我们什么都没有定义，怎么就多出来几个 Items 呢？因为在 Zabbix 的 Web scenario 中，对于每一个 scenario 和 step，都会自动生成 Items。对于 scenario 有以下几种。

(1) Download speed for scenario <Scenario> : 整个 scenario 的下载速度，是每一步下载速度的平均值。

(2) Failed step of scenario <Scenario> : scenario 中失败的 step 个数。

(3) Last error message of scenarios <Scenario> : 最近一次 error 返回的错误信息。

对于 step 有以下几种。

(1) Download speed for step <Step> of scenario <Scenario> : step 的下载速度。

(2) Response time for step <Step> of scenario <Scenario> : step 的响应时间。

(3) Response code for step <Step> of scenario <Scenario> : step 的返回码。

其中“<Scenario>”是不同的 Scenario 的名字，“<Step>”是不同的 step 的名字。

已经都设置完了，看下成果吧。查看这个数据有两个方法：一是和其他 Items 一样，在 Latest Data 中看，另一个是 Web monitoring 特有的，在菜单栏中的“Monitoring”→“Web”中，所显示的信息如图 9-7 所示。

图 9-7 显示的 Items，不能像其他 Items 一样设置属性，Zabbix 会保存 Web scenario 类型的 Items 的 30 天 history 和 90 天 trends。当然，设置了监控，肯定还是要报警的，对于这些 Items，我们在设置 Trigger 的时候和其他 Item 是一样的，如图 9-8 所示。

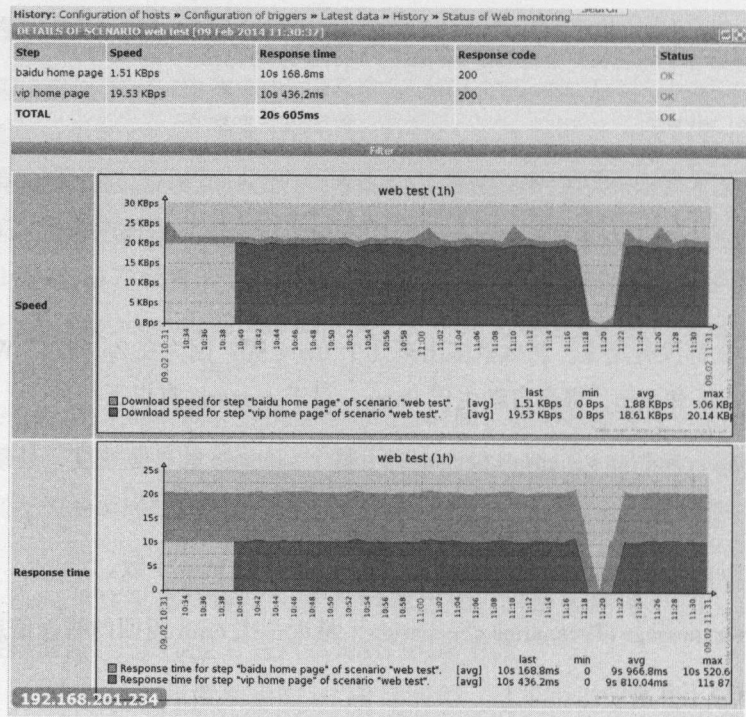


图9-7

Items - Google Chrome

192.168.201.234/zabbix/popup.php

Name	Key
Download speed for scenario "web test".	web.test.in[web test,bps]
Download speed for step "baidu home page" of scenario "web test".	web.test.in[web test,baidu home page,bps]
Download speed for step "vip home page" of scenario "web test".	web.test.in[web test,vip home page,bps]
echoA	systemm.run["echo A"]
echoB	systemm.run["echo B"]
echoC	systemm.run["echo C"]
Failed step of scenario "web test".	web.test.fail[web test]
hostname	system.hostname
Last error message of scenario "web test".	web.test.error[web test]
Response code for step "vip home page" of scenario "web test".	web.test.rspcode[web test,vip home page]
Response code for step "baidu home page" of scenario "web test".	web.test.rspcode[web test,baidu home page]
Response time for step "baidu home page" of scenario "web test".	web.test.time[web test,baidu home page,r]
Response time for step "vip home page" of scenario "web test".	web.test.time[web test,vip home page,res]
test1	web.test.in["web test",bps]

192.168.201.234

图9-8

第 10 章

Zabbix前端界面

在前面的章节中，我们简单介绍了 Zabbix 前端界面的几个部分，相信读者对 Zabbix 已经有了足够的了解。这一章，我们会统一把 Zabbix 前端界面的所有部分进行一个梳理。一方面是为了将前端介绍清楚，一方面是要让大家查缺补漏，也就是前端界面介绍到哪一部分，大家就应该能回想起这一部分大致是干什么用的，如果忘记了，再翻过去查阅前面几章的内容。希望大家在结束这一章的学习后，能够对 Zabbix 的每一个方面都有更深的印象。

为了使叙述更为清晰，笔者把一级菜单称为“板块”，一共有这些板块：“Monitoring”、“Inventory”、“Reports”、“Configuration”、“Administration”。二级菜单称为“栏目”，比如“Monitoring”板块有“Dashboard”、“Overview”、“Web”、“Latest Data”、“Triggers”、“Events”、“Graphs”、“Screens”、“Maps”、“Discovery”、“IT services”这些栏目。

10.1 Monitoring板块

Monitoring 的作用是显示监控数据，它下面的栏目就是不同的数据，比如“Triggers”等。这个板块应该是最常用的了，Guest 也可以访问。

10.1.1 Dashboard栏目

Dashboard 的中文意思是“仪表盘”，就跟飞机、汽车的仪表盘一样，它显示的是整体性能数据。在 Zabbix 中，Dashboard 栏目显示的是 Zabbix 整体的情况，包括 Zabbix 目前监控的性能

数据，哪些 Host 或者 Host group 出问题了，问题有多严重之类。Dashboard 的目的就是能够通过一个页面，可以使得用户看到整个 Zabbix 的状况，包括 Zabbix 自身和它监控的服务器的状况。

屏幕左侧的三个模块是“Favourite graphs”、“Favourite screens”、“Favourite maps”，它们是显示我们收藏的东西的，默认是空的。以“Favourite graphs”为例，它的最下方有“Graphs>>”按钮，单击后可以进入 Monitoring 板块的 Graphs 栏目。

在 Dashboard 栏目中，每一个模块都可以进行拖动和折叠，右上角的箭头标志是折叠，在它边上的按钮是菜单键，单击后可以编辑项目。按住菜单按钮后可以拖动模块的位置，如图 10-1 所示。

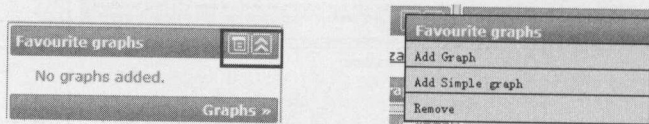


图 10-1

以“Favourite graphs”为例，单击菜单按钮，出现如图 10-1 所示选项。“Remove”用来删除已经在“Favourite graphs”中的 Graphs。“Add Graph”和“Add Simple graph”中的 Graph 指的是自定义的 Graph，比如想把 CPU 负载和空余内存放在一个 Graph 中，这个 Graph 就是普通 Graph，比如图 10-2 所示的“CPU jumps”。

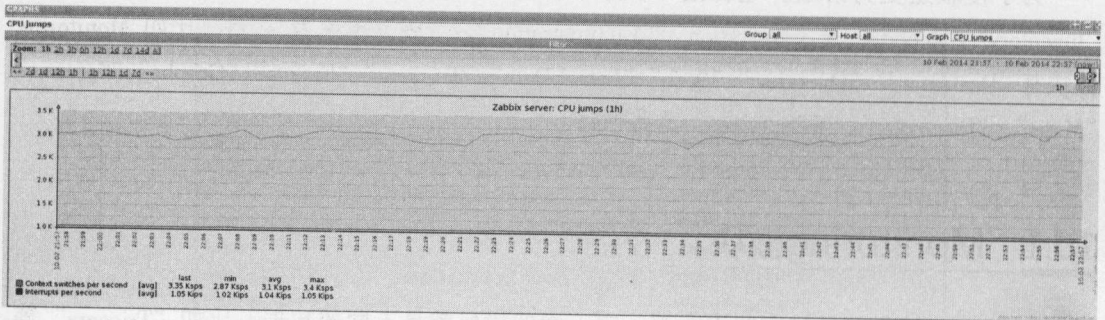


图 10-2

而 Simple graph 指的是任意一个 Item 自带的 Graphs。它们的区别不大，绘制 Graph 的时候，Zabbix 使用的是 chart.php，绘制 Simple graph 的时候，使用的是 history.php，这个在地址栏的 URL 就可以看到。单击“Add Graph”后就可以选择需要的 Graph 了。“Add Simple graph”与其类似。

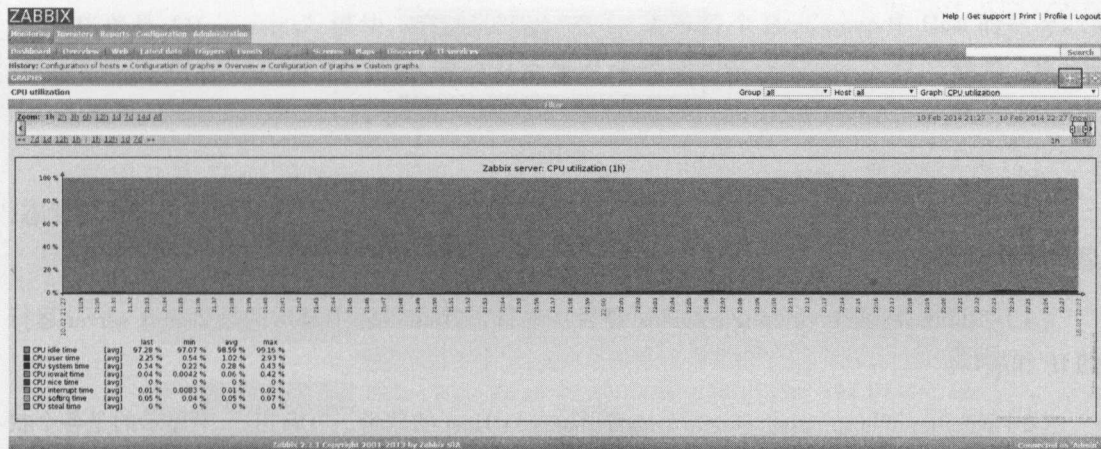


图10-3

另外一种方式，是在看 Graph（Simple graph 一样）的时候，在屏幕右上部分找到“+”号，单击，“+”号就变成“-”号了，就是从“Favourite graphs”中删除。如图 10-4 所示。

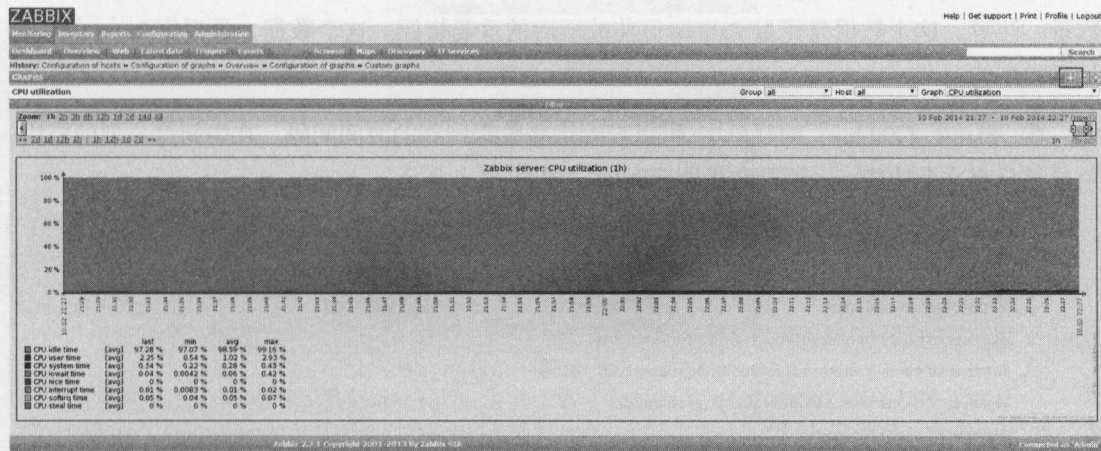


图10-4

单击以后，在“Favourite graphs”中就有了刚刚添加的 Graph 了，显示的名称形如 {HOSTNAME}:{GRAPH NAME}，单击就可以进入对应的 Graph 了。如图 10-5 所示。这个“Favourite graphs”是和账号绑定的，即不同的人看到的是不同的。

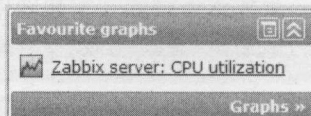


图10-5

Favourite 系列介绍完了，接下来看“Status of Zabbix”模块，它显示的是 Zabbix 的状态，显示的数据如下。

(1) Zabbix server is running : Zabbix 是否在运行，“Details”中显示的是 Zabbix server 运行的 IP 和端口。

(2) Number of hosts (items/items/triggers/users) : Host 的数量，“Details”中显示的是各个状态的 Host 数量,显示的顺序就是“Parameter”中在“Number of hosts”括号内的“monitored”、“not monitored”和“templates”，分别表示监控中的 Host、没有监控的 Host 和 Template 数量。下面都是这种表示方式。

(3) “Required server performance, new values per second” : Zabbix 每秒需要处理的新数据的数量。注意，这个数据是衡量 Zabbix 目前的压力的重要指标。这个数据，是根据 items 的数量计算出来的。

在“Status of Zabbix”的菜单中,可以设置这些数据刷新的时间间隔。在中间这一列的模块,菜单都只有这个功能——调整数据刷新时间，如图 10-6 所示。

Status of Zabbix		
Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (monitored/not monitored/templates)	52	11 / 0 / 41
Number of items (monitored/disabled/not supported)	80	68 / 0 / 12
Number of triggers (enabled/disabled) [problem/ok]	45	45 / 0 [1 / 44]
Number of users (online)	2	1
Required server performance, new values per second	1.46	-
Updated: 23:05:43		

图10-6

在“Status of Zabbix”下面的是“System status”模块，显示的是各个 Host group 的 Triggers 的状态。出问题的会标红显示。

接着是“Host status”模块，它的粒度是 Host，表示各个 Host group 中 Host 的状态。如果有 Trigger 在 problem 状态，那么这个 Host 就认为是“With problems”，反之就是“Without problems”。

图 10-7 是“Last 20 issues”模块，这里显示最近出问题的 Triggers。把鼠标放在 Issue 一列中可以看到与这个 Trigger 相关的 Events。

Issue	Last change		Age		
Zabbix discoverer processes more than 75% busy	22 Jan 2014 22:10:35	19d 1h 19m			
	Time	Status	Duration	Age	Ack
9:47	22 Jan 2014 22:10:35	PROBLEM	19d 1h 19m	19d 1h 19m	No
0	12 Jan 2014 13:54:35	OK	10d 8h 16m	29d 9h 35m	No

图 10-7

单击“Host”按钮，可以执行一些指令，或者直接查看 Host 的相关信息，如图 10-8 所示。

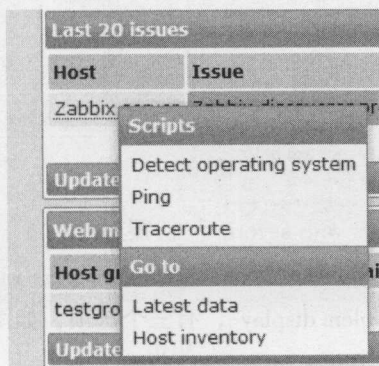


图 10-8

单击“Last change”中的时间，可以跳转到对应的 Event。也可以单击“Ack”中的“No”进行 Ack 操作。

剩下两个模块就比较简单了，一个是“Web monitoring”，另一个是“Discovery status”。“Web monitoring”显示的是各个 Host group 中 Web monitoring 的情况。“Discovery status”显示的是每一个 Discovery rule 的状况。

好了，每一个模块都介绍完了。在中间这部分，是显示所有的 Host group 的内容的。大家想想，如果有很多的 Host 和 Host group，这个界面还能看吗？可能有成百上千行，这就会和设计这个

Dashboard 的初衷就差远了，因为我们根本无法在一个屏幕清晰地看出 Zabbix 目前监控的状态。幸好，Zabbix 考虑到了这个问题。

在 Dashboard 界面右上角找到如图 10-9 所示两个按钮。



图10-9

右侧的是“全屏”按钮，左侧的是配置 Dashboard 的选项，它可以解决 Host 太多从而使 Dashboard 显示内容过多的问题。单击进去可以看到，这是一个过滤的功能。默认是“Disable”的，单击“Disabled”就可以使其变为“Enabled”的了，在 Dashboard 上，这个标志会变为红色，让我们知道当前显示的可能不是所有的 Host group 的数据。

可以使用的过滤条件并不多，一个是过滤想要显示或要隐藏的 Host group，使用方法就是将“Host groups”选择为“Selected”，然后在下面进行配置，如图 10-10 所示。

图10-10

后面的“Hosts”和“Triggers with severity”就很好理解了，“Show host in maintenance”选择显示的时候是否包含在 maintenance 状态的 Host。“Triggers with severity”可以选择显示哪些等级的 Triggers。而最后的“Problem display”，有三个选项，如图 10-11 所示。

图10-11

它表示的意思是 Problem 的 Trigger 的显示方式，“All”是指 ack 和没有 ack 的放在一起显示，“Separated”是分开显示，“Unacknowledged only”是只显示没有 ack 的。我们看图吧，在 10-12 的两幅图中，上图是“All”，下图是“Separated”。“Unacknowledged”的只是把 ack 的去掉了，这里就不上图了。

System status

Host group	Disaster	High	Average	Warning	Information	Not classified
Discovered hosts	0	0	0	0	0	0
testgroup	0	0	1	0	0	0
Zabbix servers	0	0	1	0	0	0

Updated: 23:52:21

Host status

Host group	Without problems	With problems	Total
Discovered hosts	9	0	9
testgroup	1	1	2
Zabbix servers	0	1	1

Updated: 23:52:21

System status

Host group	Disaster	High	Average	Warning	Information	Not classified
Discovered hosts	0	0	0	0	0	0
testgroup	0	0	1 of 1	0	0	0
Zabbix servers	0	0	1 of 1	0	0	0

Updated: 23:52:41

Host status

Host group	Without problems	With problems	Total
Discovered hosts	9	0	9
testgroup	1	1 of 1	2
Zabbix servers	0	1 of 1	1

Updated: 23:52:41

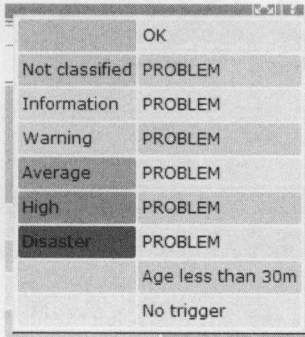
图 10-12

10.1.2 Overview 栏目

Overview 栏目，是一个非常整体的数据展示。它显示的是某个 Host group 或某个 Application 的 Triggers 状态或者是 Items 数据。

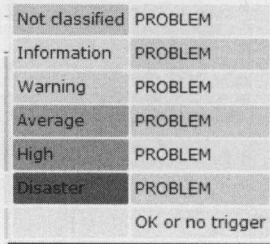
这个界面非常简单，右上角的问号处，鼠标移动上去后可以看到一些说明。图 10-13 和图 10-14 分别是在查看 Triggers 和 Data 的时候，问号的提示。

在左上角，可以显示标题栏是在上面还是在左边，如图 10-15 所示。



	OK
Not classified	PROBLEM
Information	PROBLEM
Warning	PROBLEM
Average	PROBLEM
High	PROBLEM
Disaster	PROBLEM
	Age less than 30m
	No trigger

图10-13



Not classified	PROBLEM
Information	PROBLEM
Warning	PROBLEM
Average	PROBLEM
High	PROBLEM
Disaster	PROBLEM
	OK or no trigger

图10-14

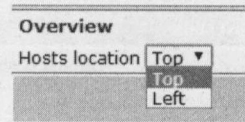


图10-15

页面最好要做一下筛选再使用，否则数据会非常多。我个人不太使用这个页面，因为数据量太大，根本看不清。

10.1.3 Web栏目

这个栏目显示的是 Web Scenario 中定义的 Web 监控的内容，单击某个项目，可以查看详细信息，如图 10-16 所示。

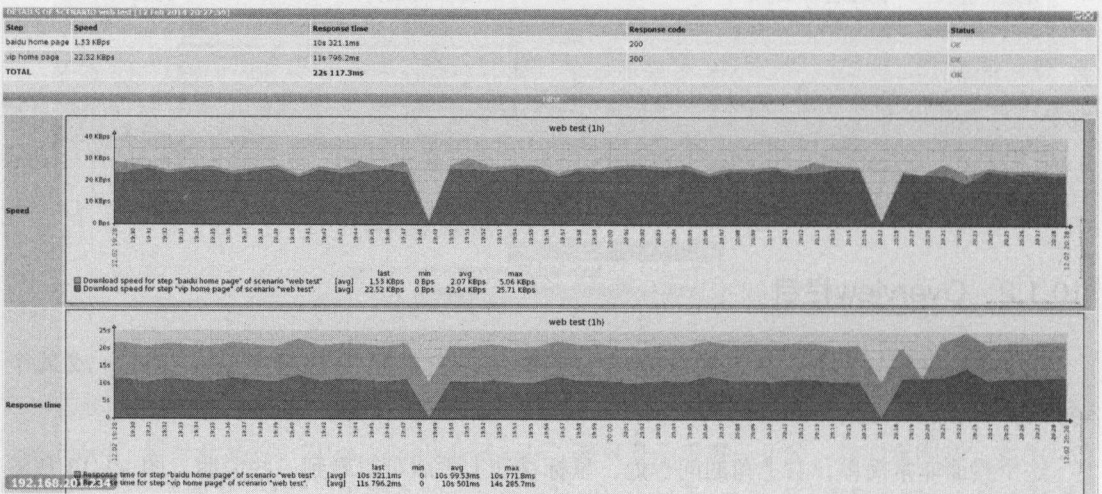


图10-16

10.1.4 Latest data 栏目

这个栏目显示的是 Zabbix 监控到的最新数据，像 Overview 一样，这里显示的数据非常多，在右上角可以进行筛选，选择需要的 Host group 和 Host。需要注意的是中间这个 Filter，如图 10-17 所示。

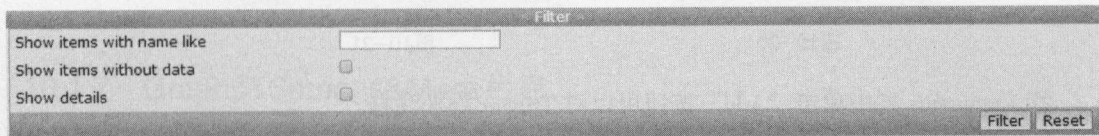


图 10-17

“Show items with name like” 很容易理解，即只显示当前页面中包含指定字符串的 Items，不支持正则表达式。

“Show items without data” 是显示没有数据的 Items，如 10-18 图所示。

Zabbix busy http poller processes, in %	13 Feb 2014 20:54:39	4.01 %	-31.55 %
Zabbix busy icmp pinger processes, in %	13 Feb 2014 20:54:40	0 %	-0.01 %
Zabbix busy jmx poller processes, in %	-	-	-
Zabbix busy java poller processes, in %	-	-	-
Zabbix busy node watcher processes, in %	-	-	-
Zabbix busy poller processes, in %	13 Feb 2014 20:54:44	0.02 %	-

图 10-18

“Show details” 使得我们可以查看详细信息，如图 10-19 所示。

Name +	Interval	History	Trends	Type	Last check	Last value	Change		Error
CPU (13 items)									
Context switches per second <small>system.cpu.switches</small>	60	7	365	Zabbix agent	13 Feb 2014 20:56:18	1.08 Kaps	+32 sps	Graph	OK
CPU idle time <small>system.cpu.idle</small>	60	7	365	Zabbix agent	13 Feb 2014 20:56:19	98.45 %	-0.15 %	Graph	OK
CPU interrupt time <small>system.cpu.interrupts</small>	60	7	365	Zabbix agent	13 Feb 2014 20:56:20	0.01 %	-	Graph	OK
CPU iowait time <small>system.cpu.iowait</small>	60	7	365	Zabbix agent	13 Feb 2014 20:56:21	0.0042 %	-	Graph	OK

图 10-19

这个时候单击 Name 中的 key，可以打开 Items 的配置界面。

10.1.5 Triggers 栏目

进入 Triggers 的界面，单击“Last change”中的时间可以跳转到 Trigger 的 Event 页面，单击“Acknowledge”可以进行 Ack 操作。单击 Host 和 Name 分别是图 10-20 和图 10-21 所示的菜单：

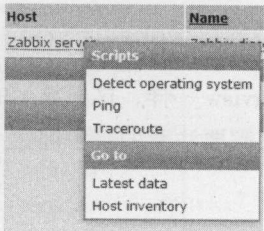


图 10-20

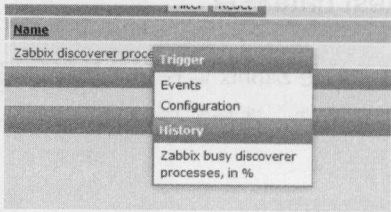


图 10-21

在 Comments 列中单击“Add”按钮可以对 Trigger 添加注释。

在 Triggers 栏目中，也有“Filter”，它可以根据设置显示需要的 Triggers。图 10-22 所示 Filter 界面。

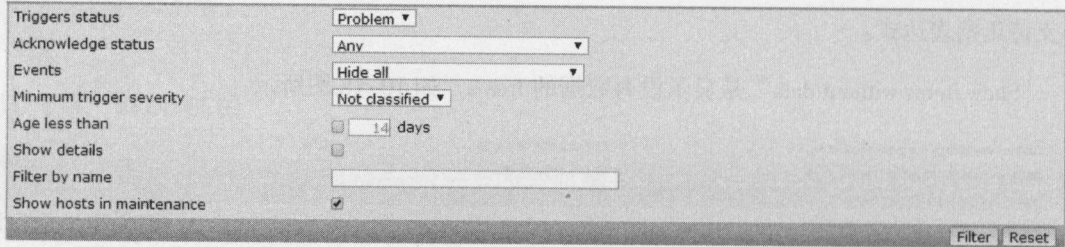


图 10-22

这里对“Acknowledge status”的选项进行说明。

- (1) Any：包含任意状态的 Trigger 都会显示。
- (2) With unacknowledged events：包含没有被 ack 的 event 的 trigger 会显示。
- (3) With last event unacknowledged：最新的一个 event 没有被 ack 的 trigger 会显示。

10.1.6 Events栏目

Events 栏目显示的是所有 Zabbix 中的 Events，右上角的“Export to CSV”可以将目前显示的 Events 导出到 CSV 文件，这个对于汇总报告分析是比较有用的。还有就是不起眼的“Source”下拉列表，我们可以在其中选择是显示 Events 还是 Discovey 的记录。如图 10-23 所示。

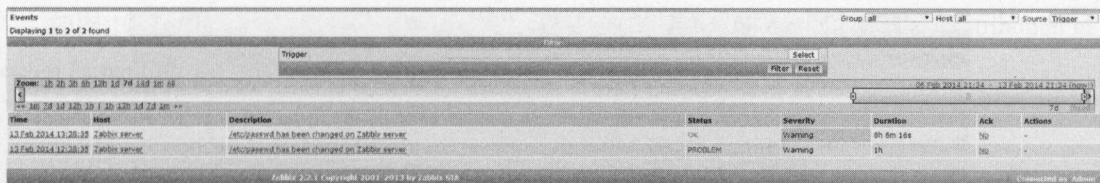


图 10-23

10.1.7 Graphs&Screens&Maps栏目

把这三个放一起讲，一是由于它们在页面上非常简单，二是它们都属于数据可视化部分。这三个大家比较熟悉，这里不做赘述，大家从界面上进入就能看到相应的 Graph、Screen 和 Map。

10.2 Inventory 板块

Inventory 板块显示的是 Zabbix 中 Host 的一些硬件信息，它由两个栏目组成，一个是 Overview，一个是 Host。在 Overview 中，可以根据某一个 Inventory 的属性来筛选需要的设备。Hosts 栏目显示的是某个 Host group 中 Host 的主要 Inventory 信息。在 Filter 中，可以进一步过滤符合条件的 Host。

10.3 Reports 板块

监控系统的数据对于运维和研发来说是非常重要的。在 Reports 板块中，Zabbix 提供了简单的对于 Zabbix 本身和监控的数据的统计报表。

它有 4 个栏目，分别是“Status of Zabbix”、“Availability report”、“Triggers top 100”和“Bar reports”。

“Status of Zabbix”和在 Dashboard 中显示的是一样的，这里不再赘述。如图 10-24 所示。

Status of Zabbix		
Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (monitored/not monitored/templates)	52	11 / 0 / 41
Number of items (monitored/disabled/not supported)	80	68 / 0 / 12
Number of triggers (enabled/disabled) [problem/ok]	45	45 / 0 [1 / 44]
Number of users (online)	2	1
Required server performance, new values per second	1.46	-
Updated: 23:05:43		

图 10-24

“Availability report”显示的是 Triggers 一段时间的 OK 次数所占的百分比。在右上角的“Mode”可以选择是按照 Host 来查看，还是按照 Template 的维度来查看。单击“Filter”可以调整显示的时间，默认是 1 天。单击“Graph”列中的 Show，可以看到这个 Trigger 每一周的 OK 次数百分比。

在“Triggers top 100”中，可以看到最近状态变化（从 OK 到 PROBLEM 或从 PROBLEM 到 OK）最频繁的前 100 个 Triggers，在右上角的下拉框可以选择时间范围。

“Bar reports”是 Reports 板块下功能较多的一个栏目。它的作用是使我们可以查看一段时间范围内的某个 Item 的值的变化的情况，图 10-25 显示的就是从 2 月 1 号到 2 月 16 号 Zabbix server 上的磁盘空闲空间的报告。

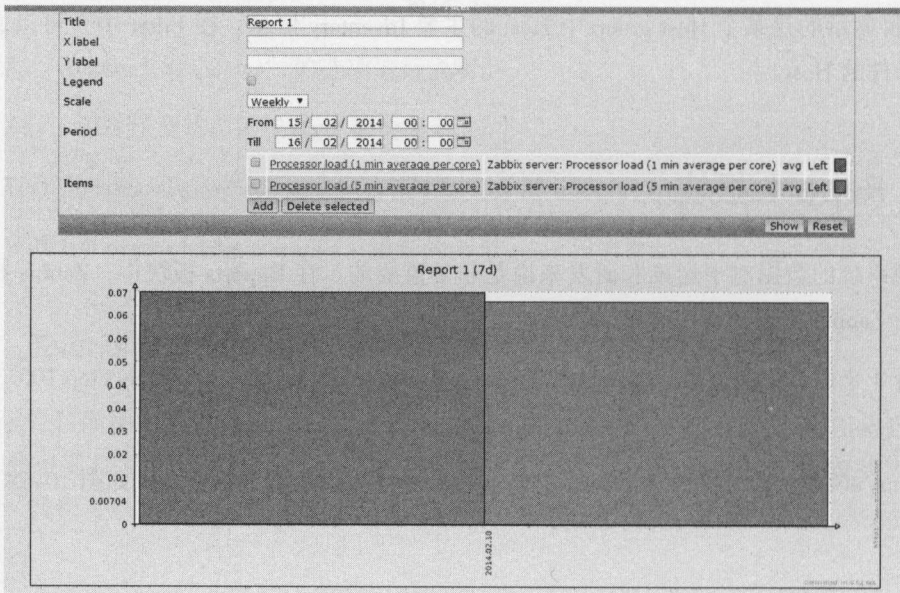


图 10-25

如图 10-26 所示，在右上角，可以选择不同的报表类型。图上是默认的“Distribution of values for multiple periods”，即“不同 Items 在同一个时间段的数据”。

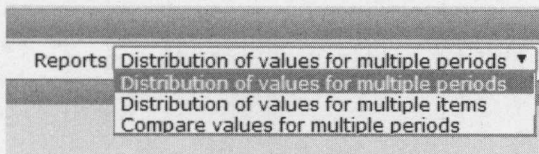


图10-26

“Distribution of values of multiple items”意为“不同时间段的不同 Items 的数据”，从图 10-27 中可以看出可以选择多个 Items 和多个时间段。

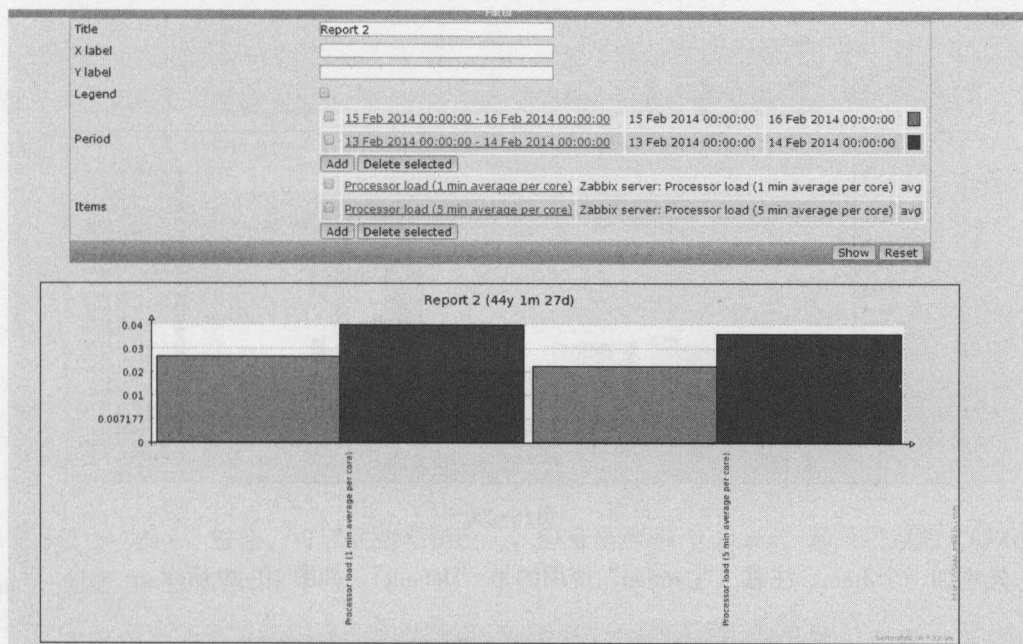


图10-27

对于“One item data comparison”，它是针对时间和 host 这个维度的，显示的是“同一个 Item，不同机器在不同时间的数据”，如图 10-28 所示。

在 Reports 板块中，有一个地方容易搞错，不知道算不算前端界面的 bug。我们一起来看看下。

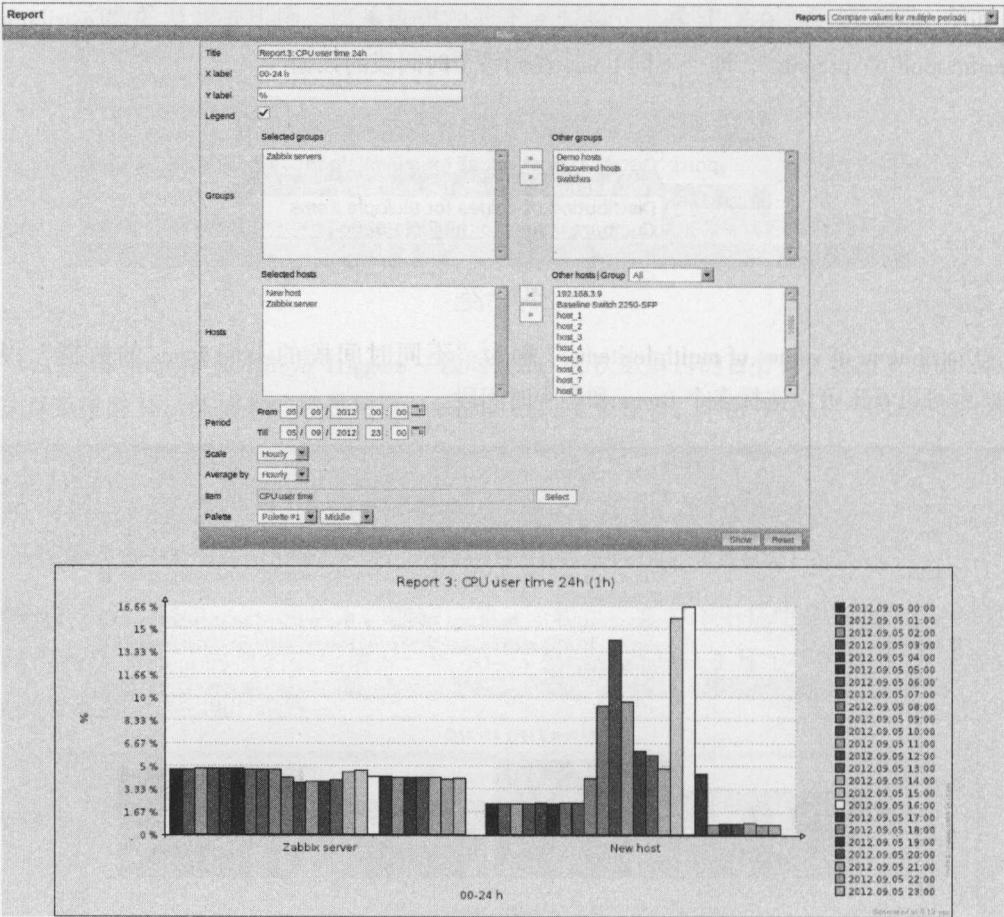


图10-28

先添加一个 Item，注意，“Caption”使用的是“Default”，如图 10-29 所示。

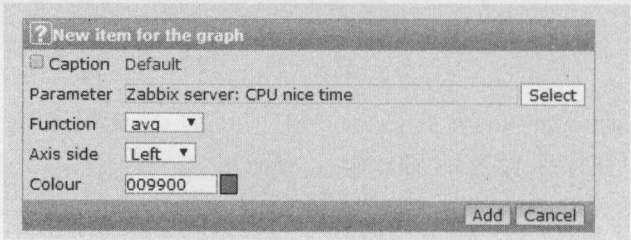


图10-29

看图 10-30 所示的界面，和笔者设置的一样。

图 10-30

这时笔者发现鼠标放在“Left”字样左侧的绿色（表示柱状的颜色）上，好像可以单击，笔者自然认为单击以后会弹出一个界面用于改变颜色。但是单击后没有任何变化。另外，单击“Items”中的“CPU nice time”，更换一个 Item，如图 10-31 所示。

图 10-31

再单击“Save”按钮，可以看到图是改了，但是显示的“Caption”却是之前的“CPU nice time”，如图 10-32 所示。

是不是很奇怪？之前选择的“Caption”明明是“Default”。回过头看看，再单击这里的“CPU nice time”就明白了，原来之前我们单击这个 Item，想换一个 Item 显示的时候，“Caption”已经不是“Default”了，而变成了写死的“CPU nice time”，所以就算换了 Item，显示的“Caption”还是之前的。解决办法就是在更改 Item 的时候，把“Caption”前面的勾去掉，变回“Default”。

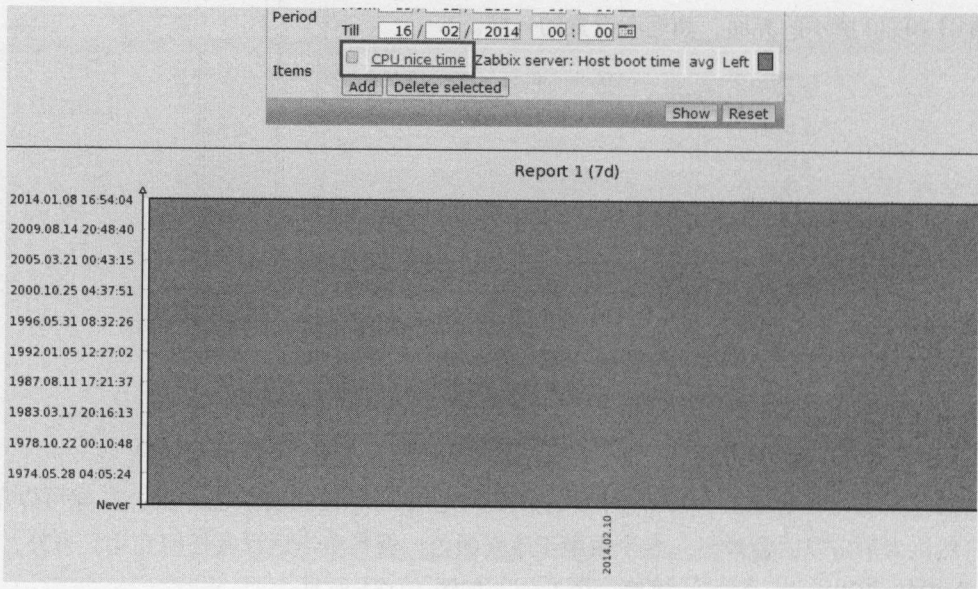


图10-32

这个设计非常奇怪，笔者怀疑这是 Zabbix 前端没有考虑这点造成的。

10.4 Configuration 板块

前面介绍了 Zabbix 里面可以进行配置的项目，而配置这些东西的界面，就在 Configuration 板块。本节主要向大家介绍界面上的内容，比如，每一列表示的是什么意思。希望大家通过这一节的学习，能够把 Configuration 板块的各项内容弄清楚。

10.4.1 Host groups 栏目

在这个栏目，能看到目前所有的 Host groups 和其中包含的 Hosts 或者是 Templates（Template 在 Zabbix 中也被认为是 Host）。看下 Host groups 栏目的界面，如图 10-33 所示是一行片断。

<input type="checkbox"/>	Name ↑	#	Members
<input type="checkbox"/>	Discovered hosts	Templates (0) Hosts (9)	127.0.0.8 , 127.0.0.9 , 127.0.0.10 , 127.0.0.2 , 127.0.0.3 , 127.0.0.4 , 127.0.0.5 , 127.0.0.6 , 127.0.0.7

图10-33

图 10-33 中,“Name”即为 Host group 的名字,“#”能够告诉我们这个 Host group 中有多少 Template、多少 Host,比如“Templatres(0)”表示这个“Discovery hosts”的 Host group 中没有 Template。“Members”列表示有多少个 Host。如果“Members”中显示的是 Templates, Templates 是灰色的。如图 10-34 所示。

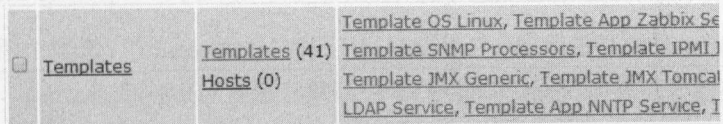


图10-34

界面右上角有“Create host group”,单击后能创建新的 Host group。左下角有个一个操作按钮,这个是在几乎所有 Configuration 板块中都能看到的按钮,但每个地方的功能有些许不同。要使用就选中需要的 Host group,在拉框中选择对应的操作,然后单击“Go”按钮,在 Go 字样后面的括号中,显示的是选中的 Host group 的个数。在这里,有以下三个选项。

- (1) Enable selected : 将选中的 Host group 中的所有 Host 变为“Monitored”状态。
- (2) Disable selected : 将选中的 Host group 中的所有 Host 变为“Not monitored”状态。
- (3) Delete selected : 删除选中的 Host group。

10.4.2 Template 栏目

Templates 栏目显示的是与 Templates 相关的信息。界面上显示了 Templates 中有多少 Application、多少 Item 等信息。“Linked templates”和“Linked to”分别显示 template 之间的关联关系。右上角的“Group”下拉框可以选择显示某个 host group 中的 template。在整个页面的左下角,也有一个操作按钮如图 10-35 所示,具体的操作如下。

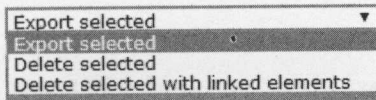


图10-35

- (1) Export selected : 导出选中的 Template。
- (2) Delete selected : 删除选中的 Template, 但不会取消 Items 等的关联。

(3) Delete selected with linked elements : 删除选中的 Template, 并取消 Items 等的关联。

对于这个导出功能, 就是将设定好的 Template 导出为 XML 文件, 从而在其他地方使用。界面右上角有一个 “Import” 按钮, 就是相应的导入。

笔者截取了自己创建的 XML 文件的开头, 大家可以看一下 Template 导出的 XML 的格式, 代码如下。

```
<?xml version=" 1.0" encoding=" UTF-8" ?>
<zabbix_export>
<version>2.0</version>
<date>2014-02-16T12:07:52Z</date>
<groups>
  <group>
    <name>Templates</name>
  </group>
</groups>
```

10.4.3 Hosts栏目

Hosts 栏目中规中矩, 显示 Hosts 的具体信息, 经过前面对其他栏目的介绍, 相信大家已经对这种界面很熟悉了。每一列都可以点进去查看详细信息。Hosts 栏目最后一列的 “Availability” 应该是大家第一次接触到的, 这里做一下说明。它一共有 4 列, 分别表示如下 4 个方面的可用性。

(1) Agent

(2) SNMP

(3) JMX

(4) IPMI

这里以 Agent 的可用性为例展开介绍, 其他三个与它类似, 读者可以按此思路来研究。

Agent 可用性一共有三种状态: “可用”、“不可用”、“未知”, 在界面上显示的图标如图 10-36 所示。



图 10-36

第 1 个是绿色的，表示“可用”；第 2 个是红色的，表示“不可用”；第 3 个是灰色的，表示“未知”。Agent 可用性表示的是 Agent 是否可用，那么 Zabbix 是怎样计算这个可用性的呢？

首先要理解两个概念：“Unreachable”和“Unavailable”，它们是针对 Host 的。Zabbix server 会每隔一段时间（UnreachableDelay 变量，默认为 15 秒）向 Agent 发送一个检查，如果没有正常返回，会认为这个 Host 是 Unreachable 状态。然后在 Zabbix server 的 Log 中记录形如“Zabbix agent item [system.cpu.load[percpu,avg1]] on host [New host] failed: first network error, wait for 15 seconds”的日志。Timeout 这个设置，对于“Unreachable”检查也有影响。比如 UnreachableDelay 是 20 秒，Timeout 是 30 秒，那么在一次检查后，要隔 50 秒才会进行下一次检查。假设现在是 00:00:00，开始执行一次检查，一直到 00:00:30，Zabbix server 才会认为这次检查超时，然后再等待 UnreachableDelay 的 20 秒，直到 00:00:50 才会执行下一次检查。

当进入 Unreachable 后，Zabbix server 会进行计时，当一个 Host 持续一段时间（由 UnreachablePeriod 定义，默认是 45 秒）后，会认为 Host 是 Unavailable 状态，即“不可用”状态。注意，UnreachablePeriod 一定要设置为 UnreachableDelay 的数倍以上，否则，一旦触发了 Unreachable 状态，就没有补救的机会，而会进入 Unavailable 状态。这样会太敏感，不推荐如此设置。

当 Host 进入 Unavailable 状态后，Zabbix server 端会有日志记录：

temporarily disabling Zabbix agent checks on host [New host]: host unavailable

这时，前端界面显示的就是红色的标志了，鼠标移动到上面之后，还能看到具体的问题。这里还有个变量——“UnavailableDelay”，它的意思是当 Host 在 Unavailable 状态中，要多久进行一次检查，看连接是否恢复。默认是 1 分钟。当检查发现连接恢复后，在 Zabbix server 中会有如下的日志：

enabling Zabbix agent checks on host [New host]: host became available

下面研究一下 Zabbix 获取 Item 数据时调用的 get_values 方法中的一段代码。

```
switch (errcodes[i])
```

```
{
```

```
    case SUCCEED:
```

```

        case NOTSUPPORTED:
        case AGENT_ERROR:
            activate_host(&items[i], &timespec);
            break;
        case NETWORK_ERROR:
        case GATEWAY_ERROR:
            deactivate_host(&items[i], &timespec, results[i].msg);
            break;
        default:
            zbx_error("unknown response code returned: %d", errcodes[i]);
            assert(0);
    }

```

可以发现,当 Items 的返回码是“SUCCEED”、“NOTSUPPORTED”和“AGENT_ERROR”时,Zabbix 都认为这个 Host 是可用的。而对于“NETWORK_ERROR”和“GATEWAY_ERROR”,则会认为是“不可用”的。

刚开始接触时,可能理解比较困难,多看几次,就会好了。Zabbix 是用 C 语言写成的,虽然写起来麻烦,但都能看得懂。我们并不是要去改写代码(当然能改写最好),重要的是理解 Zabbix 处理问题的思路。

10.4.4 Maintenance 栏目

这个界面可以配置 Maintenance 相关。State 列有以下三种显示。

(1) Approaching : 将要激活的 Host。

(2) Active : 活动的 Host。

(3) Expired : 没有激活的 Host。

10.4.5 其他

Configuration 板块的其他内容在之前报警配置部分已经进行过说明,读者可前往查看,此处不再详述。

10.5 Administration 板块

相比 Monitoring 和 Configuration 板块,大家可能会对 Administration 板块比较陌生。Administration 板块主要是与 Zabbix 管理相关的一些配置,涉及用户、分布式架构、权限和审计相关内容。一般使用 Zabbix 的人来说不太会接触,但对于负责 Zabbix 的人来说是非常重要的。

10.5.1 General 栏目

这里是对 Zabbix 整体的一些配置,有不同的项目,通过右上角的下拉框选择需要配置的项目。

下面来看一下 General 栏目的配置信息。

1. GUI

(1) Default theme : 前端界面的主题,默认的是 Original blue。

笔者个人还是比较偏好 Classic 的,贴近 iOS7 的扁平化风格,而且也比较干净。后面那几个实在是太科幻或者另类了。当我们选择了某个主题后,要重新登录,才能看到效果。

(2) Dropdown first entry : 在显示某些项目的时候,选择是默认显示全部,还是默认不显示。如选择为“None”,进入“Monitoring”→“Latest Data”后,会发现没有显示任何内容,而右上角 Group 和 Host 的地方变成如图 10-37 所示。

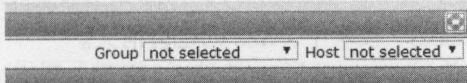


图10-37

选择为“All”时如图 10-38 所示,就会默认显示所有的数据。

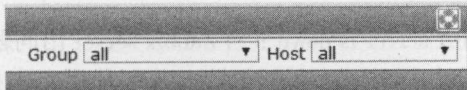


图10-38

旁边的“remember selected”的作用是记忆选择框的内容。比如在 Group 和 Host 中选择了某个 Group 和某个 Host,离开这个页面再返回时,下拉框里的内容是否还是上次离开时的内容,就是由“remember selected”配置决定的。

当 Zabbix 监控内容很多时，“remember selected”非常有用。每次打开 Latest Data 都要把所有 Group 的所有 Host 数据全部罗列一遍，需要很长时间，但其实可能只需要看某一台 Host 的数据。建议将“Dropdown first entry”设置为“None”，不勾选“remember selected”。

(3) Search/Filter elements limit：在搜索或者使用过滤器的时候，显示的条目数。比如设置为 10，那么在“Configuration”→“Templates”中可以看到如图 10-39 所示的字样。

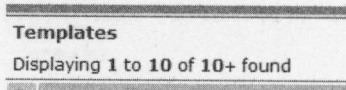


图 10-39

这表示的是现在只显示了 10 个，但总数量是超过 10 个的。在过滤器中，也只显示前 10 个项目。

(4) Max count of elements to show inside table cell：在表格中，一个单元格显示多少元素。设置为 1 时，看“Configuration”→“Templates”中显示有哪些 Host 属于一个 Template，如图 10-40 所示。

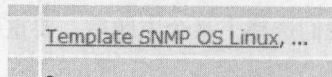


图 10-40

(5) Enable event acknowledges: Event 的 ack 在 Zabbix 前端中是否有效。在“Monitoring”→“Events”中，如果不勾选这个项目（默认是勾选），那么 Ack 列会消失。在“Monitoring”→“Dashboard”上的“Last 20 issues”模块中，Ack 列也会消失。

(6) Show events not older than (in days)：在 Trigger 状态显示多久的 Event，默认是 7 天。

(7) Max count of events per trigger to show：每一个 Trigger 最多显示多少个 Event，默认是 100。

(8) Show warning if Zabbix server is down：当 Zabbix server 出问题时，是否一直在浏览器顶部显示一条信息提示用户。

2. Housekeeper

Housekeeper 是 Zabbix 的一个特性，它的作用是删除 Zabbix 数据库中很久以前的数据，具体作法是针对不同 item 的类型设定保存时间的长短。大家有疑问的可能是 Override item history

period 和 Override item trend period，二者可以简单理解为 Zabbix 保存的 item 的历史数据。

不知道大家记不记得，在设置 Item 的时候，有两个参数是配置保存 history 和 trends 时间长短的，如图 10-41 所示。

Keep history (in days)

Keep trends (in days)

图 10-41

“Override item history period”的意思是，在“Administration”→“General”设置的保存 history 的时间，是否要覆盖 Item 自己设置的保存 history 的时间。

3. Images

在这里可以配置 Zabbix 中使用的图片，也可以上传自己的图片。注意，上传的图片要小于 1024 × 1024，或者 1MB，这个参数由 ZBX_MAX_IMAGE_SIZE 来设置。如果上传的图片没有超过这个设置却显示上传失败，并且使用的是 MySQL，需要将 MySQL 的“max_allowed_packet”参数调大。这个参数对于 MySQL 来说，定义了单个网络传输包最多能传输的数据大小。在上传图片时，这个图片会存储进 MySQL。如果图片超过了“max_allowed_packet”，则无法存储。

4. Icon Mapping

Icon Mapping 是在 Maps 中有效的。当一个 Host 的某个 Inventory 满足某个 Icon Mapping 关系后，Host 在 Maps 中显示的就是 Icon Mapping 设置的图标。

比方，在 Icon Mapping 设置了如图 10-42 所示的 Mapping 关系。

Name:

Mappings	Inventory field	Expression	Icon	
\$ 1:	OS	Cloud	Cloud (128)	<input type="button" value="Remove"/>
Add				
Default			Cloud (24)	<input type="button" value="Add"/>

图 10-42

然后将某个 Host 的 OS Inventory 设置为 Cloud，那么在 Maps 中的显示如图 10-43 所示。

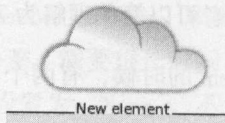


图10-43

如果一个 Host 同时满足多个 Mapping 关系,那么按照 Mapping 的先后顺序,写在前面的生效。假如 Icon Mapping 如图 10-44 所示。

Name:

Mappings

Inventory field	Expression	Icon
# 1: OS	Cloud	Cloud (128) Remove
# 2: Type	Linux	Crypto-router (24) Remove
Add		
Default		Cloud (24)

图10-44

而 Host 的 OS Inventory=Cloud 并且 Type Inventory=Linux,那么最后显示的会是 OS 对应的图标。

5. Regular expressions

在 Expression 中可以输入正则表达式。

Zabbix 在传统的正则表达式上又封装了一层,这个不是很好理解,并且 Zabbix 的官方文档也非常简略。笔者的理解是,一个 Regular expression 表达式由多个正则式和判断逻辑组成,可以使用 “@” 加上 Regular expression 的名字的方式在 Zabbix 的其他地方引用。要记住的一点是,在平时使用正则表达式的时候,一般是用来查找,即返回的是字符串。比如用 “^ext” 去匹配 “ext3” 返回的就是 “ext”。而在 Zabbix 中, Regular expression 返回的是 True 或者是 False。

下面先看一个 Regular expression, 如图 10-45 所示。

Name:

Expressions

Expression	Expression type	Case sensitive	
ext4	Result is TRUE	Yes	Edit Remove
^ext	Result is TRUE	Yes	Edit Remove
Add			

图10-45

图中设置了两个正则表达式,分别是 “ext4” 和 “^ext”。前面讲过,在 Zabbix 中一个 Regular expression 的返回值是 True 或 False,那么 True 或 False 是怎么计算出来的呢?它是根

据每一个 Expression 的返回值,最后做“与”操作完成的。图中的第一条 Expression,表示:“当字符串能够不区分大小写地被正则式‘ext4’匹配时,那么就返回 True”。第二条 Expression 的意思也显而易见了,这里不做赘述。下面,切换到 Test 标签来测试一下,如图 10-46 所示。

Result	Expression	Expression type	Result
	ext4	Result is TRUE	FALSE
	^ext	Result is TRUE	TRUE
	Combined result		FALSE

图 10-46

在这里可以看到每一条 Expression 对于测试数据的返回结果,以及整个 Regular expression 的返回结果。

接下来看看不同的 Expression type 的作用,为了操作简单,只设置一个 Expression。“Result is TRUE”和“Result is False”是最简单的。

(1) Result is TRUE: 当正则表达式能够匹配到给定字符串时, Expression 返回 True。

(2) Result is FALSE: 当正则表达式不能匹配到给定字符串时, Expression 返回 True。

(3) Character string included: 当字符串中包含正则表达式时,选择“Character string included”后, Expression 中的字符串不再被认为是正则表达式,只是作为字符串和给定的字符串进行比较。举个例子,当 Expression 是“ext.*”,给定的字符串是“ext4”时,返回值为 False。下面和“Character”相关的都是如此。

(4) Any character string included: 在这个选项中,多了个参数“Delimiter”,表示分隔符。它的判断逻辑稍微复杂一些,会根据给定的分隔符,分隔给定的字符串。分隔后的字符串有任何一部分包含 Expression 设定的字符串,就返回 True。

(5) Character string not included: 给定的字符串不包含 Expression 设定的字符串,就返回 True。

Regular expression 在什么地方用呢,举个例子,在设置某个 Host 或者 Template 的 Discovery rule 的地方,有一个 Filter,如图 10-47 所示。

Filter Macro Regexp @testRegularExpressi

图 10-47

这里就可以引用创建的 `testRegularExpression`，它会根据返回的 `True` 或者 `False` 决定下一步动作。

6. Macro

定义 Zabbix 系统级的宏。

7. Value mapping

配置好后，在 Item 设置中，可以选择需要的 Value mapping，当 Item 配置了一个 Value mapping 后，会根据返回值和 Value mapping 中的映射关系来显示值。比如配置了“0->OK”这个 Value mapping，那么当 Item 返回这个值以后，就会看到“OK”而不是“0”，这是为了让使用的人更好地阅读。

8. Working time

定义工作时间，它在 Graph 中以不同的背景来显示。比如 Working time 是 9 点到 18 点，那么一个 Graph 如图 10-48 所示。

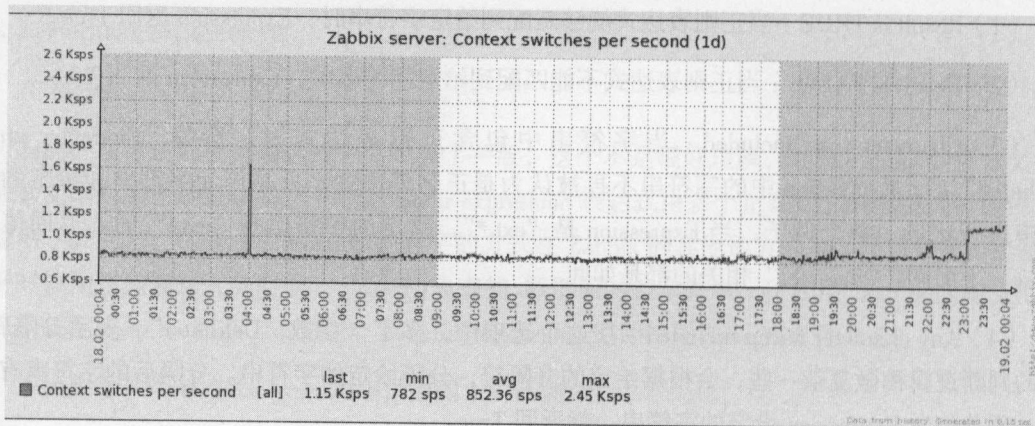


图 10-48

这里可以定义多组时间。每一组时间的格式如下：

`d-d, hh:mm-hh:mm`

“d-d”表示的意思是从星期几到星期几，后面的表示从什么时间开始到什么时间结束。“d”可以从 1 到 7，表示星期一到星期日。“hh”是 24 小时制，从 00 到 24，“mm”从 00 到 59。多组时间之间用分号分隔。

比如“1-5,09:00-18:00;6-7,10:00-16:00”表示的是“星期一到星期五的 9 点到 18 点及星期六和星期日的 10 点到 16 点”。

9. Trigger severities

自定义 Trigger 的等级名称和颜色。强烈建议不要修改，因为改了还要改各个语言文件中的翻译。

10. Trigger displaying options

显示 Trigger 相关的一些设置，Blinking 可以设置某个状态的 event 是否闪烁。

11. Other

这里是其他的一些设置，但并不是没用的，只是没法将它们放到前面的任何一个项目中，才在这里讲解。

(1) Refresh unsupported items (in sec) : 一些 Items 会因为 user parameter 设置错误或者自身的配置问题而变成 unsupported 状态。Zabbix 会定期将这些 Item 变为 active 状态。这里设定的就是这个时间间隔。如果设置为 0，则不会将 unsupported 的 Item 变为 active。这个参数对 proxy 无效，假如 Proxy 每隔 10 分钟检查监控的服务器上的 unsupported 的 Item，那这个 10 分钟的时间间隔是不能设置的。

(2) Group for discovered hosts : 通过“network discovery”和“agent auto-registration”添加的 Host，会自动添加到这里设置的 Host group。默认为“Discovery hosts”。

(3) User group for database down message : 当 Zabbix 后端数据库发生问题的时候，会发送通知给 User group。监控数据库的工作是由一个叫做“Database watchdog”的 Zabbix server 进程来处理的。当数据库失去连接的时候，Zabbix server 进程不会退出，它会继续等待，直到恢复了数据库连接。

(4) Log unmatched SNMP traps : 如果 SNMP trap 有问题，将其记录在日志中。

10.5.2 DM 栏目

DM 是“Distributed Monitoring”的缩写，在这里能够配置 Zabbix 分布式监控相关的内容。

10.5.3 Authentication栏目

Zabbix 的用户认证除了使用自带的登录系统外，还可以使用常见的 LDAP 和 HTTP 形式。在公司中，最常用的应该是和 LDAP 集成的。下面先看看如何为 Zabbix 配置 LDAP 登录，如图 10-49 所示。

Default authentication: ☒ Internal ☐ LDAP ☐ HTTP

LDAP host:

Port:

Base DN:

Search attribute:

Bind DN:

Bind password:

Test authentication: [must be a valid LDAP user]

Login:

User password:

图10-49

(1) LDAP Host : LDAP 服务器的地址。比如 `ldap://ldap.zabbix.com`，如果是 LDAPS (“S”表示 secure，类似于 HTTP 和 HTTPS)，就是 `ldaps://ldap.zabbix.com`。

(2) Port : LDAP 默认是 389，LDAPS 默认是 636。

(3) Base DN: “DN”是“Distinguished Name”的缩写，即 LDAP 的一个唯一性的名字引用。根据不同的 LDAP 来设置，对于 OpenLDAP 来说，是“`ou=Users,ou=system`”；对于 Microsoft Active Directory 来说，是“`DC=company,DC=com`”。

(4) Search attribute : 在 LDAP 中搜索的属性，对于 LDAP 使用“uid”，对于 Microsoft Active Directory 使用“`sAMAccountName`”。

(5) Bind DN : 绑定 DN，用来在 LDAP 服务器上搜索 LDAP 账户的账户。对于 OpenLDAP，可以使用“`uid=ldap_search,ou=system`”这种形式，对于 Microsoft Active Directory，可以使用“`CN=ldap_search,OU=user_group,DC=company,DC=com`”这种形式。Bind DN 一定要填写，匿名用户是不允许的。

(6) Bind password : 和 Bind DN 一组的密码。

(7) Test authentication : 使用测试用户登录 LDAP 的返回信息。

(8) Login : 用来测试 LDAP 设置是否成功的用户名,是不能更改的,默认是用当前登录 Zabbix 的用户名来测试。需要注意的是,如果当前用户无法通过 LDAP 认证,那么是无法使用 LDAP 来管理 Zabbix 登录的。

(9) User password : 用来测试的用户的密码。

总的来说,Zabbix 配置 LDAP 还是比较简单的。但是在配置的界面中,Zabbix 使用的参数名字比较容易让人误解。如果是我来设计这个页面,我会把 Test authentication、Login 和 User password 放在另外一个框中,并把 Login 改成 Username to test LDAP, User password 改成 Password to test LDAP, Test authentication 改成 Test Result。

当设置为使用 LDAP 的时候,原有的一些 Zabbix 用户还可以继续登录,只要属于拥有访问前端权限的用户组就行了,即拥有“frontend access”权限。

HTTP 是使用基于 Apache 的 HTTP 认证。在启用之后,所有认证都会通过 Apache,而不是 Zabbix,并且之前的 session 全部无效。就算拥有“frontend access”权限,也无法登录 Zabbix。

10.5.4 Users 栏目

Users 栏目设置与 Zabbix 用户相关的内容。进入这个栏目时,显示的是 User group 的信息,下面介绍下 Frontend access 和 Debug mode 的含义。

(1) Frontend access : 当 User 属于的 User group 有 Frontend access 权限时,就算使用了 LDAP 认证,也可以使用该 User 访问 Zabbix。要达到这样的效果,需要将 Frontend access 设置为 Internal。Frontend access 一共有三个选项可以选择。

① System default : 这个是默认的,即 Zabbix 在“Administration”→“Authentication”中设置使用什么登录,那么就只能用这种方式登录。

② Internal : 无论 Zabbix 设置了什么登录方式,都可以使用 User 登录。

③ Disabled : 禁止该用户访问 Zabbix 前端。

(2) Debug mode : 打开 Zabbix 前端的 Debug 模式。

将 Debug mode 打开后,在界面右上角单击“Debug”按钮,会在屏幕下方显示 Zabbix 在载入当前 PHP 时进行的操作。

Debug 信息有两种，一种是 Zabbix API 调用的参数和结果，比如：

```

usergroup->get [usergrps.php:89]
Parameters:
Array
(
    [usrgrpsids] => 7
    [output] => extend
)
Result:
Array
(
    [0] => Array
        (
            [usrgrpid] => 7
            [name] => Zabbix administrators
            [gui_access] => 0
            [users_status] => 0
            [debug_mode] => 1
        )
    )
)

```

这个是调用了 usergroup 的 get 方法，从 usergrps.php:89 可以知道执行的是 usergrps.php 中的第 89 行，运行的参数是 Parameters 中的内容，结果是 Result 中的内容。

另一种信息就是执行的 SQL：

SQL(0.000864) : SET NAMES utf8

usergrps.php:22 → require_once () → ZBase->run () → ZBase->initDB () → DBconnect () → DBexecute () in /var/www/html/zabbix/include/db.inc.php:61

这条日志表示的意思是，“SET NAMES utf8”这条 SQL 执行了 0.000864 秒，是在 usergrps.php 的 22 行调用，后面是 PHP 方法调用链，最后是在 db.inc.php 的 61 行执行了。

User group 讲完了，下面讲 User 部分。它显示的信息比较简单，就是一些 User 本身的属性。稍微说明下“Login”这列。有两种值：“OK”和“Blocked”。当一个用户连续 5 次使用错

误密码尝试登录后,就会变为“Blocked”状态,防止其暴力破解密码。对于已经变成“Blocked”的用户,可以单击 Login 列中的 Blocked, 将其解锁。

10.5.5 Media types 栏目

Used in actions : 如果在某个 Action 中设置了直接发给某个 Media, 这一列就会显示 Action 的名字, 比如在 Action 中进行如图 10-50 所示的设置。

Default operation step duration: 3600 (minimum 60 seconds)

Steps	Details	Start in	Duration (sec)	Action
1	Send message to user groups: Zabbix administrators via Email	Immediately	Default	Edit Remove
1	Run remote commands on hosts: HostA	Immediately	Default	Edit Remove

Operation details

Step: From: 1 To: 1 (0 - infinitely) Step duration: 0 (minimum 60 seconds, 0 - use action default)

Operation type: Send message

Send to User groups: User group: Zabbix administrators Action: [Remove](#)

Send to Users: User: [Add](#) Action: [Add](#)

Send only to: Email

Default message: ☒

Conditions: Label: Name: Action: [New](#)

[Update](#) [Cancel](#)

图10-50

那么在这一列的设置如图 10-51 所示。

Displaying 1 to 3 of 3 found

<input type="checkbox"/>	Name ↑	Type	Status	Used in actions
<input type="checkbox"/>	Email	Email	Enabled	Report problems to Zabbix administrators
<input type="checkbox"/>	Jabber	Jabber	Enabled	-
<input type="checkbox"/>	SMS	SMS	Enabled	-

图10-51

10.5.6 Scripts 栏目

在 Monitoring 板块中, 有 Host 出现的地方, 单击“Host”按钮后, 都可以对 Host 远程执行一些命令, 如图 10-52 所示。

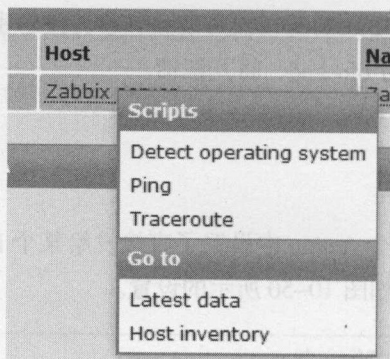


图10-52

这里可以有三个操作：“Detect operating system”、“Ping”和“Traceroute”。单击“Ping”按钮，就会弹出一个窗口，显示从 Zabbix server 去 Ping 这台 Host 的结果，如图 10-53 所示。

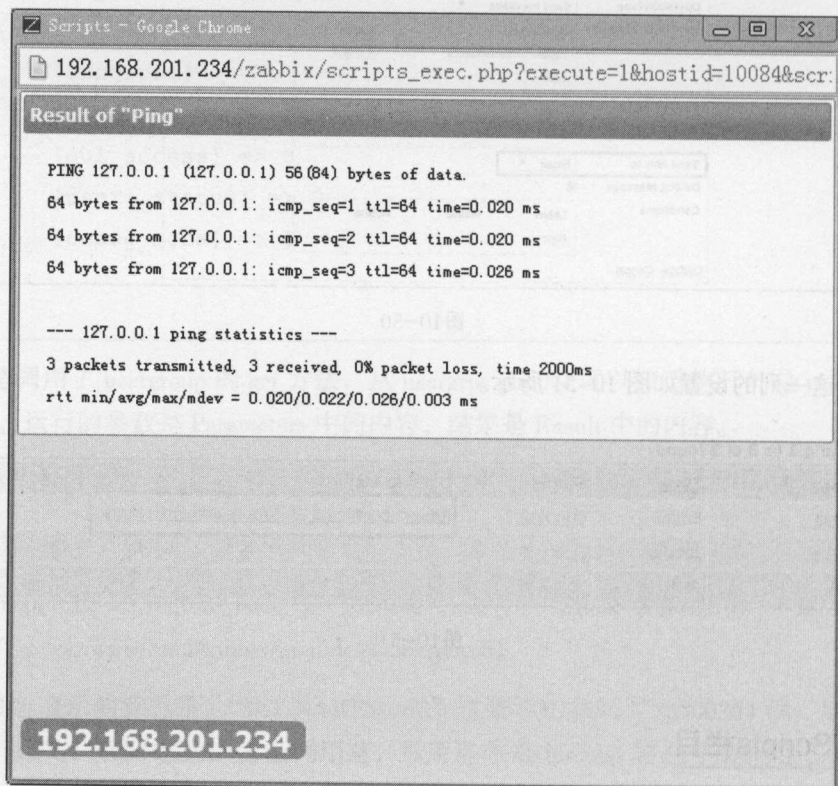


图10-53

这个就是现在 Scripts 栏目的作用，可以看到这三个命令在 Scripts 栏目的定义如图 10-54 所示（因为篇幅受限，图中只截取了部分列）。

Name	Type	Execute on	Commands
<input type="checkbox"/> Detect operating system	Script	Server	sudo /usr/bin/nmap -O {HOST.CONN} 2>&1
<input type="checkbox"/> Ping	Script	Server	/bin/ping -c 3 {HOST.CONN} 2>&1
<input type="checkbox"/> Traceroute	Script	Server	/usr/bin/traceroute {HOST.CONN} 2>&1

图 10-54

其中，

- Name：脚本的名字，单击“Host”按钮后显示。
- Type：脚本的类型，可以是 Script 和 IPMI。Script 就是 shell 命令。
- Execute on：可以选择脚本是在 Zabbix server 上执行还是在 Zabbix agent 上执行。
- Commands：执行的命令。
- User group：哪些 User group 可以执行。
- Host group：这个脚本可以在哪些 Host group 中的 host 上执行。
- Host access：User 对于 Host 要有怎样的权限才可以执行脚本。可以选择的有“Read”和“Write”。选择 Read，即表示 User 要对 Host 有 Read 权限才可以执行命令。Write 选项与其类似。

我们随便点开一个已经存在的脚本，或者新建一个，看下一个脚本需要的配置，如图 10-55 所示。

其中各项的含义如下。

- Name：脚本的名字。这里有个高级的地方，就是可以定义层级的命令。在 Name 中输入“def/Detect operating system”，然后单击“host”按钮。

经过测试，三层也是可以的，比如 Name 是“efg/def/Detect operating system”。

- Execute on：选择在 Zabbix server 还是 Zabbix agent 上运行。注意，如果选择 Agent，那么需要在 Zabbix agent 的配置文件中，将 EnableRemoteCommands 设置为 1。
- Commands：运行的命令。建议输入命令的绝对路径，防止环境变量影响执行。一般在命令中都会使用宏，比如 Zabbix 自带的脚本中使用的 {HOST.IP}。个人建议最好在宏外面套上引号，防止宏中的空格影响执行。Commands 中支持的宏有：

The screenshot shows the Zabbix configuration interface for a new item. The form is as follows:

- Name:** Detect operating system
- Type:** Script
- Execute on:** Radio buttons for Zabbix agent (unselected) and Zabbix server (selected).
- Commands:** sudo /usr/bin/nmap -O {HOST.CONN} 2>&1
- Description:** (Empty text area)
- User groups:** Zabbix administrators
- Host groups:** All
- Required host permissions:** Read
- Enable confirmation:** ☐
- Confirmation text:** (Empty text area) Test confirmation

图10-55

- {HOST.CONN}
- {HOST.IP}
- {HOST.DNS}
- {HOST.HOST}
- {HOST.NAME}
- User Macro

◎ User groups/Host groups/Required host permissions：这几个选项控制了脚本可以在哪些服务器上运行。User groups 和 Host groups 很容易理解，分别从 User 和 Host 两个角度定义。而 Required host permissions 表示的意思是只有当用户对于 Host 的权限超过我们设置的级别后，才可以运行。比如我们在这里设置了 Read，那么用户至少要对这个 Host 有 Read 权限才可以执行脚本。

◎ Enable confirmation：在执行前会弹出窗口，让你确认是否执行，对于比较危险的指令是需要的，就好像在 Linux 上 rm 也会让大家确认一样。

◎ Confirmation text : 弹出确认信息的提示语, 在这里支持以下宏。

- {HOST.HOST}
- {HOST.NAME}
- {HOST.IP}
- {HOST.DNS}
- {HOST.CONN}
- User Macro

10.5.7 Audit 栏目

Audit 栏目的作用是记录 Zabbix 中的一些动作和变更, 还有 Action 的执行情况。默认是显示前者, 比如从界面上看最新的变更就是在前面, 把一个脚本名字改了的动作, 如图 10-56 所示。

Time	User	IP	Resource	Action	ID	Description	Details
20 Feb 2014 22:40:11	Admin	192.168.80.13	Script	Updated	0		Name [efg/def/Detect operating system] id [3]

图 10-56

在 Filter 中可以选择关注的 User、Action 和 Resource。在时间的选择上, 和 Graph 是一致的。

我们设置了很多 Action, 那肯定也关心在什么时候、因为什么、执行了怎样的 Action。在 Audit 中可以看相应的信息, 如图 10-57 所示。

一个 Action 一共有如下 4 种 Status。

- (1) in progress : Action 正在进行中。
- (2) sent : 通知已经发出。
- (3) executed : 命令已经执行。
- (4) not sent : Action 还没有完成。

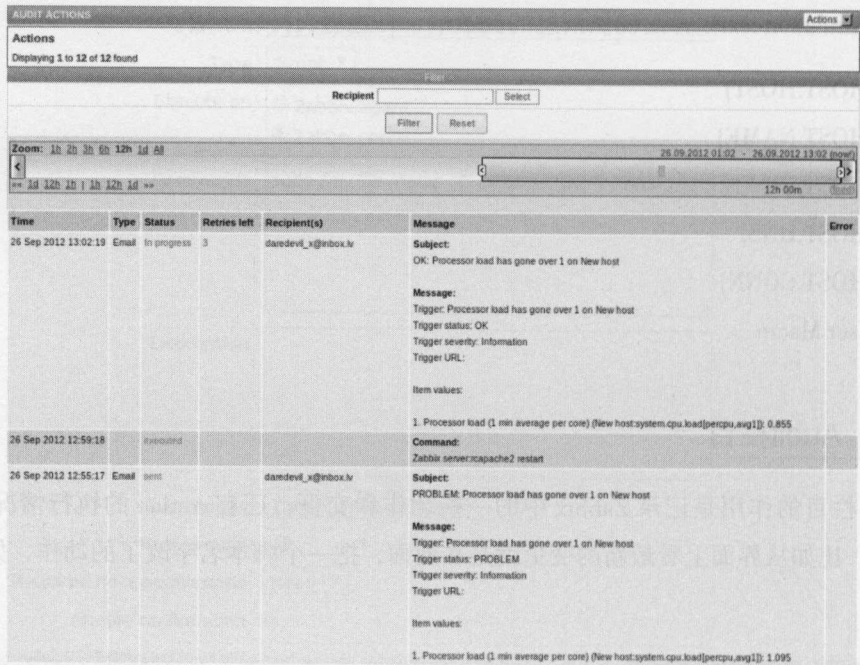


图10-57

10.5.8 Queue栏目

Producer-Queue-Consumer 模型的作用就是将消息的产生者和消息的消费者解耦。消费者不用关心消息给谁消费了，只要把消息往一个 Queue 扔就行了。Zabbix 也有类似的机制，可以理解为从 Host 收集的监控数据是扔在一个 Queue 里，Zabbix server 会把这些数据从 Queue 里拿出来处理。这样 Queue 里堆积了多少数据，是说明 Zabbix 性能是否有问题的重要标准。如果 Zabbix 非常健康，那么 Queue 里就不应该有数据堆积；反之如果 Zabbix 性能碰到了瓶颈，那么就来不及处理 Queue 里的数据，从而 Queue 里的数据就堆积起来了。

打开 Queue 栏目，可以看到目前整体的各个 Queue 的堆积情况。右上角可以切换不同的视图。
在右上角单击“Details”后可以看到目前 Queue 到底堆积了哪些数据。

10.5.9 Notification栏目

这里显示了一段时间内 Zabbix 发送通知的统计信息，在右上角可以选择 Media type 和时间。

10.5.10 Installation 栏目

这是配置 Zabbix 的地方，一般安装以后是不会进入这里的。通过配置最后生成一个配置文件，如果前端有东西需要修改，比如后端的 Zabbix 数据库变更了，只需要修改 php 目录下 conf 中的 zabbix.conf.php 就行。

10.6 前端配置

10.6.1 全局配置参数

Zabbix 的前端是使用 PHP 开发的，和其他语言一样，前端的一些配置参数是写在配置文件中的全局变量，配置文件的位置在 include/defines.inc.php 中。默认 defines.inc.php 是这样的：

```
define ('ZABBIX_VERSION',      '2.2.0');
define ('ZABBIX_API_VERSION', '2.2.0');
```

从默认的内容就可以了解如何书写这些配置了，接下来我们看看一共有哪些配置。

(1) ZBX_PERIOD_DEFAULT : Graph 中显示的时间跨度，单位是秒，默认是 1 小时，如图 10-58 所示。

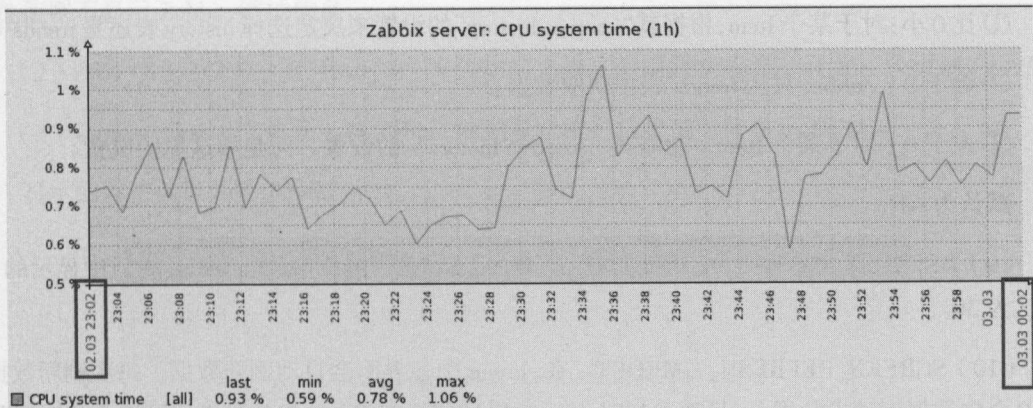


图 10-58

(2) ZBX_MIN_PERIOD : Graph 中显示时间的最短跨度，这个滑块的最小值。如图 10-59 所示。

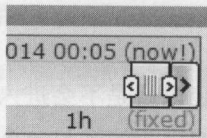


图10-59

(3) ZBX_MAX_PERIOD : Graph 中显示时间的最长跨度。

(4) GRAPH_YAXIS_SIDE_DEFAULT : Graph 中 Y 轴的位置。0 表示左侧, 1 表示右侧。默认是 0, 即我们看到的 Y 轴在左侧。

(5) ZBX_UNITS_ROUNDOff_THRESHOLD : 选择使用 ZBX_UNITS_ROUNDOff_UPPER_LIMIT 或者 ZBX_UNITS_ROUNDOff_LOWER_LIMIT 的阈值, 默认为 0.01。

(6) ZBX_UNITS_ROUNDOff_UPPER_LIMIT : 当值大于 ZBX_UNITS_ROUNDOff_THRESHOLD 时, 显示的数字在小数点后的位数, 默认为 2。

(7) ZBX_UNITS_ROUNDOff_LOWER_LIMIT : 当值小于 ZBX_UNITS_ROUNDOff_THRESHOLD 时, 显示的数字在小数点后的位数, 默认为 6。

(8) ZBX_HISTORY_DATA_UPKEEP : 在 Graph 章节中我们说过, 当需要显示的数据的时间跨度过长时, 会选择用 trends 表的数据来显示。这个“长”的阈值, 就是这里的设置。可以有三种设置。

① 比 0 小: 对于某个 Item, 根据其“keep in history”的设置来决定选择 history 表还是 trends 表。

② 等于 0 : Zabbix 只从 trends 表中获取数据。

③ 大于 0 : 对于某个 Item, 忽略其“keep in history”的设置, 而是用这里的设置。

默认为 -1。

(9) DEFAULT_LATEST_ISSUES_CNT: 在 Dashboard 板块中的 Last n issues 栏目中显示的 n, 默认为 30。

(10) SCREEN_REFRESH_TIMEOUT : 在 screen 中, 界面会自动刷新数据。每次刷新数据, PHP 会向数据库请求数据, 一旦请求超时, screen 区域就会变暗。这里就是设置了这个超时时间, 默认为 30 秒。

(11) SCREEN_REFRESH_RESPONSIVENESS : 由于 screen 在刷新数据时, 是一个一个发

送请求的，为了避免一个查询超时而堵塞了其他查询，这个参数针对每一个查询设置了超时时间。单位是秒，默认为 10。

10.6.2 前端维护状态显示

当 Zabbix 要维护的时候，可能会给用户发送邮件告知，但不是每一个用户都会注意到提示，他们在访问 Zabbix 前端的时候，会发现数据出现问题，但是不知道出了什么问题。因此，最好能在 Zabbix 前端提示用户 Zabbix 正在维护。Zabbix 确实提供了这个功能，而且使用起来非常简单。在前端目录下的 `conf/maintenance.inc.php` 中，具体内容如下。

```
// Maintenance mode
define('ZBX_DENY_GUI_ACCESS',1);
// IP range, who allowed to connect to FrontEnd
$ZBX_GUI_ACCESS_IP_RANGE = array('127.0.0.1');
// MSG showed on Warning screen!
$_REQUEST['warning_msg'] = 'We are upgrading MySQL database till 15:00.
Stay tuned...';
```

这里一共有三个可以设置的参数：

(1) `ZBX_DENY_GUI_ACCESS`: 是否在前端显示维护状态。“1”表示显示 Zabbix 维护信息，其他值则不显示 Zabbix 维护信息。

(2) `ZBX_GUI_ACCESS_IP_RANGE`: 允许访问 Zabbix 前端的 IP 范围，比如：192.168.1.1~255。

(3) `warning_msg`: 当打开了在 Zabbix 前端显示维护信息的时候显示的内容。

显示的维护信息如图 10-60 所示。

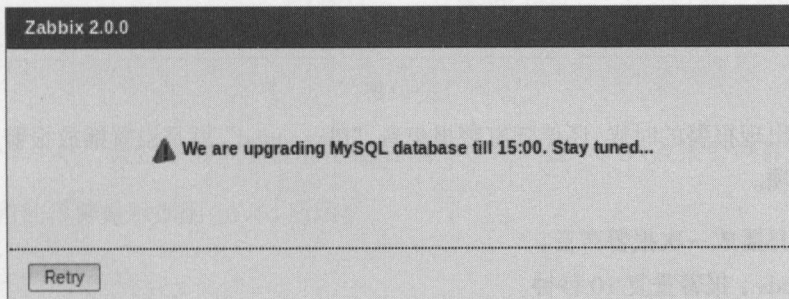



图 10-60

10.6.3 Profile设置

Profile 设置的位置在 Zabbix 前端的右上角，单击“Profile”按钮后进入。这些设置只针对该用户生效，设置界面如图 10-61 所示。



USER PROFILE: Zabbix Administrator

User Media Messaging

Password

Language

Theme

Auto-login ☒

Auto-logout (min 90 seconds)

Refresh (in seconds)

Rows per page

URL (after login)

图10-61

User 和 Media 标签的一些设置在 Administration 中都已经有了，Profile 界面只是让大家能够快速对自己当前用户进行一些配置。

这里主要介绍“Messaging”板块，有一个叫做“Global notification”的新功能。当 Zabbix 产生报警的时候，会在 PHP 前端的任何界面的右上角发出提示，告诉我们发生的问题，如图 10-62 所示。

进入 Messaging 标签，这个功能的设置如图 10-63 所示。

首先勾选“Frontend messaging”。需要说明的是“Play sound”和“Trigger severity”这两个选项，具体如下。

当右上角出现报警的时候，还伴随有警报声音，“Play sound”就是设置播放报警声音的选项，有如下三个选项。

- Once：只播放一次报警声音。
- 10 seconds：报警重复 10 秒钟。

- Message timeout: 报警声音一直重复, 直至报警在右上角消失。消失的时间是由 “Message timeout (seconds)” 定义的。

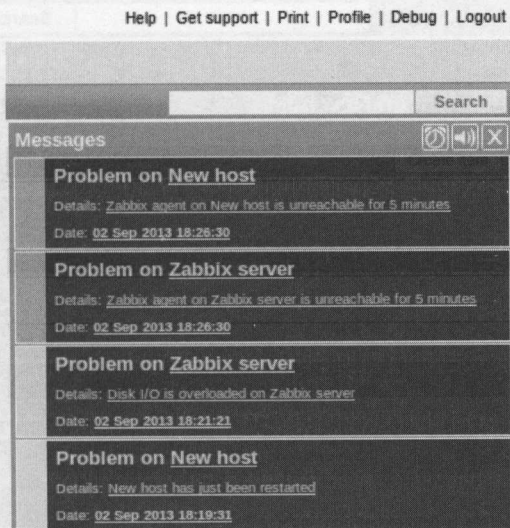


图 10-62

Frontend messaging ☐

Message timeout (seconds)

Play sound

Trigger severity			
<input checked="" type="checkbox"/> Recovery	alarm ok	▼	Play Stop
<input checked="" type="checkbox"/> Not classified	no sound	▼	Play Stop
<input checked="" type="checkbox"/> Information	alarm information	▼	Play Stop
<input checked="" type="checkbox"/> Warning	alarm warning	▼	Play Stop
<input checked="" type="checkbox"/> Average	alarm average	▼	Play Stop
<input checked="" type="checkbox"/> High	alarm high	▼	Play Stop
<input checked="" type="checkbox"/> Disaster	alarm disaster	▼	Play Stop

图 10-63

“Trigger severity” 设定了哪些等级的 Trigger 需要在前端显示, 以及报警声音是什么。

在右上角的报警显示如图 10-64 所示。

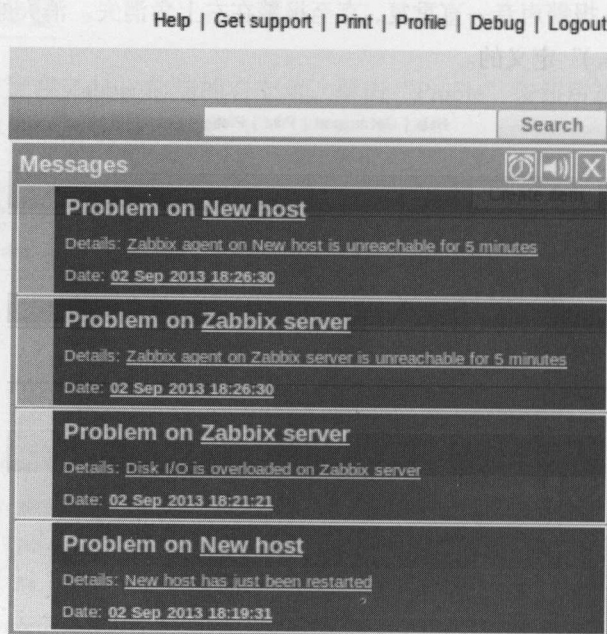


图10-64

右上角有三个按钮，从左到右分别是 Snooze、Mute/Unmute 和 Clear。Snooze 的作用是使当前的报警静音；Mute/Unmute 的作用是静音 / 取消静音所有报警；Clear 的作用是清除当前显示的报警。

10.7 全局搜索框

在本书的一开始，介绍过右上角的全局搜索框，它可以搜索 Host，在这一节中，我们讲解它的完整功能。

首先，我们能用什么来搜索什么呢？可以用 Host 中设定的 Visible name（如果没有设置，那么可以使用 hostname）、IP 地址和 DNS，可以用 Template 的名字，还可以用 Host group 的名字。

当在搜索框中输入内容的时候，Zabbix 会根据 Host 的 Hostname 进行自动补全。搜索出的结果会罗列在下方。根据输入，一共会搜索出 Host、Hosts group 和 Template。再根据权限的不同，显示的 Host 的操作也各有不同。

第 11 章

Discovery

如果要在监控系统中实现运维自动化的理念，最麻烦的就是自动化添加监控。比如监控网卡流量、监控每块硬盘的空余空间，那么就要针对每一块网卡、每一块硬盘来添加监控。经过前面几章的学习，大家应该已经知道在 Zabbix 中添加监控的具体过程了，以目前了解的情况来看，这的确是个挺麻烦的事。

幸好 Zabbix 提供了这方面的功能，叫做 Discovery，它可以去“发现”我们需要的东西，然后去添加对应的监控。

11.1 基于网络的Discovery

在本章的一开始，举了两个例子，一个是网卡的自动监控，一个是硬盘的自动监控。其实，如果服务器能够自动加入监控，岂不是更好？Zabbix 的 Discovery 确实提供了这个功能，简单来说，我们能提供 IP 范围，如果这个范围内出现了可以添加的服务器，那么就将其添加到 Zabbix 监控。

一般来说，自动添加服务器有两个步骤：“发现”和“添加监控”。“发现”这一步就是“Discovery”，而“添加监控”，就是前面讲过的“Action”。

基于网络的 Discovery，指的是基于“IP”的 Discovery，即给定 Zabbix 一个 IP 范围，对其中的每一个 IP 进行探测，如果 IP 上存在某些服务，则会触发一些 Action。Zabbix 支持下面这些服务。

(1) FTP、SSH、Web、POP3、IMAP、TCP 等服务。

(2) 从 Zabbix agent 收到的信息。

(3) 从 SNMP agent 收到的信息。

我们先总结一下: Discovery 就是会在给定的 IP 范围内, 针对每一个 IP, 检查是否存在事先设定的服务 (可以是多个服务), 并根据检查的结果, 触发 Aciton。比如, 要在 192.168.0.1~255 这个范围内, 检查是否存在 FTP 服务的服务器, 如果有, 则加入监控, 并加入 FTP Servers 这个 Host group。

在了解了这个过程之后, 可以展开来了解 Discovery 了。我们分析下前面的例子, 最关键的部分有以下两个。

(1) IP 或者服务是否存在。

(2) Action 的类型。

展开来看, Zabbix 一共可以检查如下服务。

FTP、HTTP、HTTPS、ICMP ping、IMAP、LDAP、NNTP、POP、SMTP、SMTPv1 agent、SMTPv2 agent、SMTPv3 agent、SSH、TCP、Telnet、Zabbix agent。

在每一次检查后, 会产生一个 Event, Event 一共有 Up、Down、Discovery、Lost 4 种状态, 分别对应 Host 和 Server 两种, 则一共有 8 种 Event, 具体如下。

Service Up、Service Down、Host Up、Host Down、Service Discovered、Server Lost、Host Discovered、Host Lost。

而在 Event 产生后, 可以进行的 Action 有如下几种。

(1) 发送通知

(2) 增加或删除 Hosts

(3) 开始或停止监控 Hosts

(4) 将 Host 添加到 Host group

(5) 从 Host group 中移除 Hosts

(6) 连接或取消连接模板

(7) 执行 scripts

再举个例子,可以设定在 192.168.0.1-255 这个 IP 范围内,先检查是否有 FTP 服务,如果有,则加入 FTP Servers 这个 Host group。再检查是否有 Zabbix agent 服务,如果有,则启动对这个 Host 的监控。

每一个检查(比如例子中检查 FTP 和 Zabbix agent)都是相互独立的,一个服务检查失败了不会影响其他服务的检查。

11.2 Discovery 的一个例子

本节会先从一个例子开始,让大家熟悉 Discovery 的整个流程,然后再在后面详细介绍各个参数。假设在 127.0.0.1-101 的 IP 中,将有 FTP 服务的 Host 加入 FTP Servers 这个 Host group。

首先配置 Discovery rule。

从“Configuration”→“Discovery”进入配置的首页,单击“Create rule”按钮创建一条规则,然后就可以看到配置界面了,如图 11-1 所示。

The screenshot shows the Zabbix Discovery Rule configuration window. The 'Name' field is set to 'FTPDDiscoveryRule'. 'Discovery by proxy' is set to 'No proxy'. The 'IP range' is '127.0.0.1-101'. 'Delay (in sec)' is '3600'. Under 'Checks', there is a 'New' button and a sub-form for adding a check. The sub-form has 'Check type' set to 'FTP' and 'Port range' set to '21', with 'Add' and 'Cancel' buttons. 'Device uniqueness criteria' is set to 'IP address' with a radio button. The 'Enabled' checkbox is checked. At the bottom, there are buttons for 'Save', 'Clone', 'Delete', and 'Cancel'.

图 11-1

配置了发现服务的规则,接下来要配置发现服务后的动作。新建一个以 Discovery 为 source 的 Action,如图 11-2 所示。

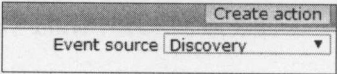


图 11-2

在 Action 的配置中，Condition 配置如图 11-3 所示。

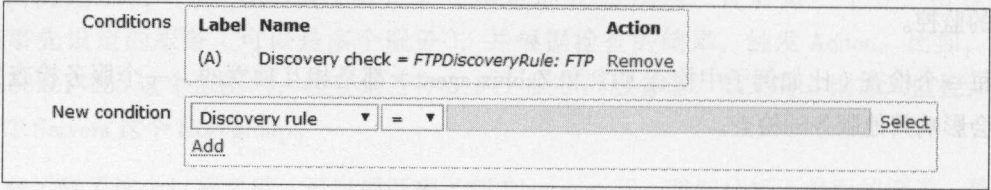


图 11-3

接下来 Operation 的配置如图 11-4 所示。

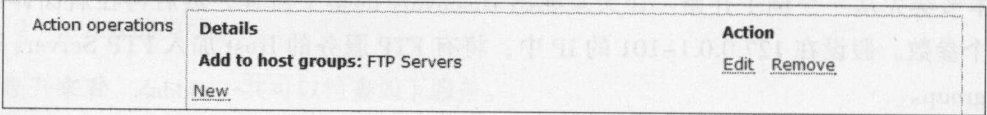


图 11-4

11.3 Discovery Rule和Discovery Action的配置

这一节中会对 Discovery Rule 的配置和 Discovery Action 的配置进行详细说明。

在 Discovery Rule 的配置中，需要解释以下几个参数。

(1) IP range : IP 地址的范围。在表示范围时，使用“-”，比如 127.0.0.1-101 就表示这个规则会对于 127.0.0.1 到 127.0.0.101 的每一个 IP 进行检查。除了这种方式，Zabbix 还支持使用子网掩码：127.0.0.1/16，对 IPv6 可以使用“/128”。

(2) Delay (in sec) : 规则检查的频率。

(3) Checks : 针对每个 IP 检查的服务，每一个服务还可以调整端口侦测的范围，比如 FTP 使用了默认的 21 端口，但某公司规定 FTP 一定要用 221 端口，那么在侦测 FTP 服务的时候就要使用 221 端口，在 Port 中更改即可，如图 11-5 所示。

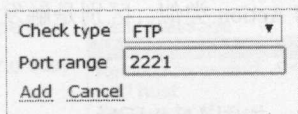


图 11-5

如果在 Check type 中选择了“Zabbix agent”，那么除了可以配置端口外，还有一个强制的参数需要配置——“Key”。在进行这个检查时，会有一个返回值，“Key”就是配置需要返回什么数据。这个配置很有用，比如可以根据返回值的不同去执行不同的操作——如果 Hostname 中包含“Linux”，则加入“Linux Servers” Host group；如果 Hostname 中包含“Windows”则加入“Windows Servers” Host group。

(4) Device uniqueness criteria：设备唯一标识，默认是使用 IP。如果在 Checks 中选择了 SNMP 相关或者是 Zabbix agent，那么这里就会有不同的选择，如图 11-6 所示。

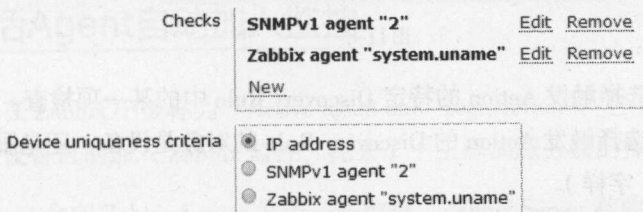


图 11-6

它表示的意思是一旦有两个设备有一样的“设备唯一标识”，那么 Zabbix 在检查后一个设备的时候就会跳过。当默认选择为 IP 时，假设 127.0.0.1 这个 IP 已经检查过了，那么在同一次检查中就会跳过这个 IP。要是如图 11-6 中一样选择“Zabbix agent ‘system.uname’”，那么就是碰到相同的 system.uname 的设备时会跳过。

在前面我们已经配置好了 Discovery Rule，那么光有规则没用，还要配置触发了这个规则以后的 Action。配置和普通的 Action 类似，我们到“Configuration”→“Actions”中，右上角的 Event source 选择“Discovery”，然后单击“Create action”来创建 Action，如图 11-7 所示。

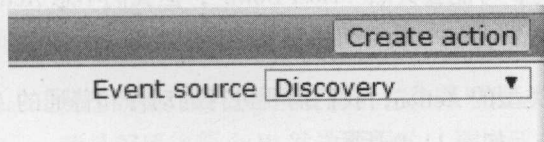


图 11-7

Action 的配置和基于 Trigger 的 Action 类似，区别是触发 Action 的条件是和 Discovery 相关，触发后的动作也是和 Discovery 相关。

触发 Action 的条件有下面几种，如图 11-8 所示。

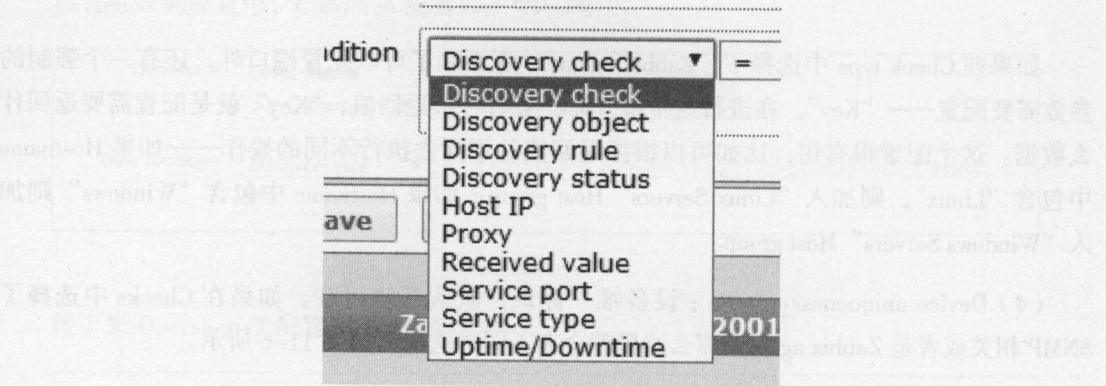


图 11-8

- Discovery check：选择触发 Action 的特定 Discovery Rule 中的某一项检查。
- Discovery object：选择触发 Action 的 Discovery Rule 的对象是设备，还是服务（下略“选择触发 Action 的”字样）。
- Discovery rule：特定的 Discovery Rule。
- Discovery status：特定的 Discovery status，比如“Up”，“Down”等。
- Host IP：Discovery Rule 的 Host 的 IP 的范围。
- Proxy：触发的 Host 是不是某个 Proxy 监控的（比如有时我们的需求是每一个 Proxy 监控的 Host 在一个 Host group 中）。
- Received value：从 Agent 收到的值是否满足一定条件。
- Service port：针对服务的 rule 是否是某个端口的。
- Service type：服务的类型。
- Uptime/Downtime：对于“Host Up”和“Service Up”来说的持续时间是否满足一定条件。比如当网络波动时，可能会突然“Host Down”，但我们不想 Action 这么敏感，可能需要配置要“Host Down”持续一段时间才触发。

对于 Discovery rule 类型的 Action，我们需要进行的动作和普通的 Action 稍有不同，Zabbix 提供了下面这几种 Action，如图 11-9 所示。

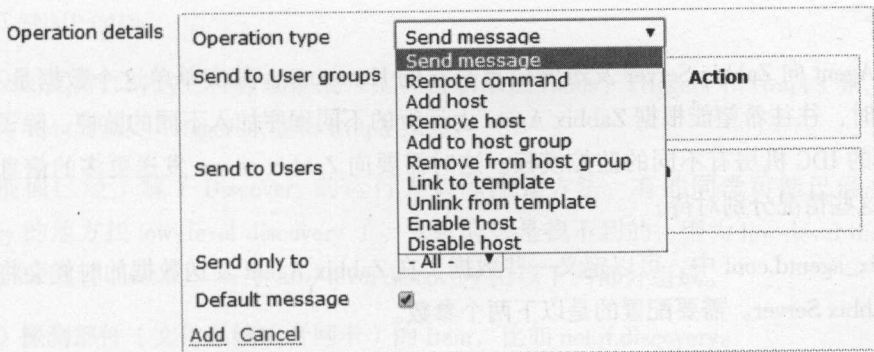


图11-9

这些选项的意思比较容易理解，从字面意思即可看出，这里就不赘述了。

11.4 存活Agent自动加入监控

这个功能在 Zabbix 中被称为“Active agent auto-registration”，它的作用是将有 Zabbix Agent 在工作的这些设备自动加入 Zabbix 监控，免去了手工添加服务器的操作。

Zabbix Agent 会向 Zabbix Server 发送一些数据，Zabbix Server 接收到这些数据以后，会根据配置的 Action 进行对应的操作，比如添加模板之类。

笔者认为这是个非常棒的功能。在 PPTV 的工作中，我们在添加一台服务器进入 Zabbix 的时候，是使用了一个叫做 spider.py 的工具，它会根据目标服务器上存在什么服务（比如 Redis 服务），按照规则加入规定的模板进行监控，从而使我们能做到一键添加监控。但这个工具有个弊端，它不能自动去发现需要添加的服务器，一定要输入 IP 才行。在后来，这个工作才由 CMDB（中央管理数据库）来完成，即在服务器上架时，就自动调用 spider.py 加入监控。

前面提到了 Agent 自动加入监控的工作原理，它是由下面几个步骤组成的。

- （1）Zabbix Agent 向 Zabbix Server 发送数据。
- （2）Zabbix Server 收到数据，调用对应的 Action 进行操作。

既然需要 Zabbix Agent 发送数据，那么必须要告诉它往哪里发。在 zabbix_agentd.conf 中有一个参数——ServerActive，就是配置往哪个 IP 发送数据的。添加到 Zabbix 监控后的 Hostname 是我们在这台服务器的 Hostname。如果在 zabbix_agentd.conf 中设置了 Hostname 属性，那么就

会以此为淮。

Zabbix Agent 向 Zabbix Server 发送的信息是它的 Hostname，其实单单这个数据是不够的。在添加监控时，往往希望能根据 Zabbix Agent 上运行的不同程序加入不同的监控，除了这个可能还要不同的 IDC 机房有不同的监控模板。这时需要向 Zabbix server 发送更多的信息才能在 Action 中对这些情况分别对待。

在 zabbix_agentd.conf 中，可以定义一些数据，在 Zabbix Agent 发送数据的时候会将这些一起发送给 Zabbix Server。需要配置的是以下两个参数。

- (1) HostMetadata：一段用户自定义的字符串。
 - (2) HostMetadataItem：使用一个 Item 的值作为发送给 Zabbix server 的 metadata
- 这两个参数只会生效一个，HostMetadata 优先生效。

如果想一次传递多个数据怎么办呢？有个剑走偏锋的方法，比如想将服务器品牌和省份两个数据发送给 Zabbix server，并且设置不同的 Action，可以进行如下操作。

- (1) 设置 HostMetadata，比如“HostMetadata=Dell Shanghai”。
- (2) 在 Action 中进行如图 11-10 所示的设置。

Label	Name	Action
(A)	Host metadata like Shanghai	Remove
(B)	Host metadata like Dell	Remove

图11-10

这样就能一次发送多个数据了。

11.5 low-level discovery

low-level discovery 的意思是“低层次的自动发现”，简称 lld。lld 并不是因为功能简单或者不重要而被称为“低层次的”，而是因为相对之前提到的服务器的自动发现，low-level discovery 是针对服务器上设备的自动发现。在 Zabbix 2.2 中，支持服务器上以下这些部件的自动发现。

- (1) 文件系统
- (2) 网卡

(3) SNMP OIDs

Zabbix 支持针对这三种自动发现来配套自动添加 Items、Triggers 和 Graphs 等, 在 lld 中它们被称为 Item 原型、Trigger 原型和 Graph 原型。

在前面已经了解了 Discovery 的运行方式和配置方法, 有的同学可能已经去之前定义 Discovery 的地方找 low-level discovery 了。大家应该是找不到的, 因为 low-level discovery 是基于 Template 或者 Host 的。一个 low-level discovery 由以下两部分组成。

(1) 探测部件 (文件系统或者网卡) 的 Item, 比如 net.if.discovery。

(2) 基于 Item 创建的 Triggers 和 Graphs。

low-level discovery 的 Item 比较特殊, 普通的 Item 返回的可能是数字、字符串等的监控数据, lld 的 Item 返回的是一个 JSON 对象, 其中包含了侦测到的部件列表。以 net.if.discovery 为例, 它会返回一些键值对, 如 “{#IFNAME}” - “lo”, 和 “{#IFNAME}” - “eth0”。这里宏的名称, 由 lld 控制。

lld 会发现很多不同的部件, 比如多块网卡, 这些变量 (网卡的名称, eth0, eth1 等) 会绑定在宏上。定义 Triggers 或者 Items 的时候使用这些宏来代替变量名就行了。针对侦测到的网卡的监控就可以是 net.if.out[{#IFNAME}]]。支持在下面这些地方使用宏:

(1) Item 原型的 name、key、SNMP OIDs、calculated item formulas、SSH and Telnet scripts、database monitor item parameters。

(2) Trigger 原型的 names、expressions。

(3) Graph 原型的 names。

下面看看 Zabbix 自带的一个 Template 的侦测文件系统的 lld。在学习这个例子后, 大家对 lld 就能很清楚地了解了。我选的是监控 Linux 的 Template OS Linux 模板中的 Mounted filesystem discovery。

单击模板后面的 “Discovery” 按钮进入 Template OS Linux 的 lld 界面, 接下来可以看到我们想要看的 lld。

在显示的界面中, 一行里除了 lld 的名字外, 显示了 Item 原型、Trigger 原型、Graph 原型和 Host 原型, 因为作为例子的 lld 没有 Host 原型, 所以笔者会在例子后面单独介绍它。“Key” 就是执行 lld 的 Item key。单击 “Name” 按钮进入 lld 的基本设置, 如图 11-11 所示。

Name

Mounted filesystem discovery

Type

Zabbix agent

Key

vfs.fs.discovery

Update interval (in sec)

3600

Flexible intervals

Interval	Period	Action
No flexible intervals defined.		

New flexible interval

Interval (in sec)

50

Period

1-7,00:00-24:00

Add

Keep lost resources period (in days)

30

Filter

Macro

{#FSTYPE}

Regexp

@File systems for disc

Description

Discovery of file systems of different types as defined in global regular expression "File systems for discovery".

Enabled

☒

Save

Clone

Delete

Cancel

图 11-11

其中，

(1) Type：和配置 Item 时的 Type 的含义一致。

(2) Key：和配置 Item 时的 Key 含义一致。

(3) Keep lost resources period (in days)：lld 侦测到的新部件的各种原型保留多少天。比如设置为 10，那么在侦测到一块磁盘后的原型会连续监控 10 天，如果在 10 天内这块盘还被侦测到了，那么这个时间会顺延。设置这个参数的目的是当服务器上的部件发生变化时，能够将不用的部件禁用。注意，这里设置为 0，并不是永远不停止，而是立刻删除。

(4) Filter：lld 的 Item 会返回 JSON 对象，其中只有一部分是我们需要的。比如，vfs.fsdiscovey 返回的 JSON 是这样的：

```
{["{#FSNAME}":"/boot", "{#FSTYPE}":"ext3"},["{#FSNAME}":"/boot", "{#FSTYPE}":"ext2"]}
```

而你只想要监控 ext3 的，那么就需要对 #FSTYPE 进行过滤，简单来说，需要 #FSTYPE 符合一些条件。这里所说的条件，在“Administration”→“General”的 Regular expressions 中配置的，它对 #FSTYPE 设置了过滤。过滤条件如图 11-12 所示。

```
1>^(btrfs|ext2|ext3|ext4|jfs|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|hfs|ntfs|fat32|zfs)$[Result is TRUE]
```

图 11-12

这里只会返回符合条件的文件系统。

lld 的创建已经完成，至此，lld 能够返回需要的文件系统了，下一步应该是根据返回的文件系统创建 items。我们接着看这个 lld 的 item 原型是怎么建立的。如图 11-13 所示。

The screenshot shows the Zabbix item prototype configuration interface. The 'Name' field is 'Free disk space on \$1'. The 'Type' is 'Zabbix agent'. The 'Key' is 'vfs.fs.size[{#FSNAME},free]' with a 'Select' button. The 'Type of information' is 'Numeric (unsigned)' and the 'Data type' is 'Decimal'. The 'Units' are 'B'. The 'Use custom multiplier' checkbox is unchecked, and the 'Update interval (in sec)' is '60'. The 'Flexible intervals' section shows a table with columns 'Interval', 'Period', and 'Action', and a row stating 'No flexible intervals defined.'. Below this, the 'New flexible interval' section has 'Interval (in sec)' set to '50' and 'Period' set to '1-7,00:00-24:00', with an 'Add' button. The 'History storage period (in days)' is '7' and the 'Trend storage period (in days)' is '365'. The 'Store value' and 'Show value' dropdowns are both set to 'As is', with a 'show value mappings' link. The 'New application' section shows a list of applications: '-None-', 'CPU', 'Filesystems' (selected), 'General', 'Memory', and 'Network interfaces'. The 'Description' field is empty. The 'Enabled' checkbox is checked.

图 11-13

从图 11-12 中可以发现，Item 原型的配置和普通的 Item 是差不多的，唯一不同的就是有的地方用了形如 {#FSNAME} 的宏来替代 lld 发现的部件的变量。比如 {#FSNAME} 就是磁盘路径。Trigger 原型和 Graph 原型都是类似的情况。

前面介绍的例子没有讲 Host 原型，可以自己新建一个试试，如图 11-14 所示。

单击 Groups、Templates、Host Inventory 会发现和普通的新建 Host 是一样的。

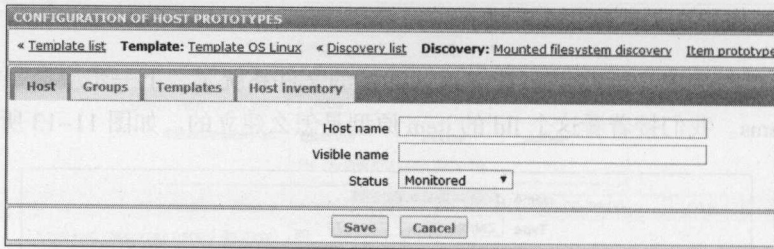


图11-14

综上所述：lld 从 Zabbix agent 中获取一些部件的信息，并根据 lld 的需要，对这些信息进行过滤。将需要的部件信息根据预先定义的原型，新建一些 Items、Triggers 等。

Zabbix 内置了一些 lld 的 Item，比如 vfs.fs.discovery 和 net.if.discovery。其实还可以自己写 lld 的 Item，只需要返回值符合规范即可。返回的 JSON 需要有一些键值对，具体格式如下。

```
{
  "data" : [

    { "{#FSNAME}" : " \\", "{#FSTYPE}" : " rootfs" },
    { "{#FSNAME}" : " \sys", "{#FSTYPE}" : " sysfs" },
    { "{#FSNAME}" : " \proc", "{#FSTYPE}" : " proc" },
    { "{#FSNAME}" : " \dev", "{#FSTYPE}" : " devtmpfs" },
    { "{#FSNAME}" : " \dev\pts", "{#FSTYPE}" : " devpts" },
    { "{#FSNAME}" : " \\", "{#FSTYPE}" : " ext3" },
    { "{#FSNAME}" : " \lib\init\rw", "{#FSTYPE}" : " tmpfs" },
    { "{#FSNAME}" : " \dev\shm", "{#FSTYPE}" : " tmpfs" },
    { "{#FSNAME}" : " \home", "{#FSTYPE}" : " ext3" },
    { "{#FSNAME}" : " \tmp", "{#FSTYPE}" : " ext3" },
    { "{#FSNAME}" : " \usr", "{#FSTYPE}" : " ext3" },
    { "{#FSNAME}" : " \var", "{#FSTYPE}" : " ext3" },
    { "{#FSNAME}" : " \sys\fs\fuse\connections", "{#FSTYPE}" : " fusectl" }
  ]
}
```

第12章

Zabbix API



第三部分 进阶篇

- ★ 第12章 Zabbix API
- ★ 第13章 Zabbix 分布式监控
- ★ 第14章 Zabbix 系统优化
- ★ 第15章 轻量级日志监控应用

第 12 章

Zabbix API

API 是 Zabbix 非常强大的功能，通过它能真正将 Zabbix 和其他系统串联到一起，使之成为整个运维体系中的一员。

Zabbix API 是一个 JSON-RPC 的 API，通过 HTTP 请求。它提供了几乎所有 Zabbix 的功能，比如更新 Item、添加 Host 监控等。

在这一章中，主要介绍 Zabbix API 的使用方法、常见的用途，以及它在运维自动化中的功能。

12.1 Zabbix API POST 参数

下面看一个最简单的例子，可以直接在 shell 中运行：

```
curl -i -X POST -H 'Content-Type: application/json' -d '{"jsonrpc":  
"2.0","method":"user.login","params":{"user":"Admin","password":"zabbix"},"a  
uth": null,"id":0}' http://192.168.122.103/zabbix/api_jsonrpc.php
```

能得到以下返回值：

```
{"jsonrpc": "2.0", "result": "3144f0e443ead877cf9ae58e659d3446", "id": 0}
```

result 即请求的返回值，针对每一个用户登录，都会有唯一的这一串数字，在之后的所有和 API 的操作中，都要带上这串数字，id 是根据 post 请求的 id 值的返回。“id”字段，是为了

标识这次返回值是针对哪一次请求的。因为如果同时发送多个查询，且没有 id，就会分不清哪个返回值是哪一次请求的结果。

来看下面这条命令：

```
{
  "jsonrpc" : "2.0",
  "method" : "user.login",
  "params" : {
    "user" : "Admin",
    "password" : "zabbix"
  },
  "id" : 1
}
```

其中的“jsonrpc”和“id”是所有 API 都有的，“method”表示这次 API 请求的动作。在 Zabbix API 中，“user.login”的方法由两部分组成：“user”为资源，“login”为方法。更新 item 就类似“item.update”这样。

接下来看 params 里面的一些“key-value”，有登录需要的“user”和“password”，即发起“user.login”需要的两个参数。

综上，Zabbix API 的 POST 参数里，“method”字段说明了发起请求的方法，“params”里有发送给 Zabbix API 的参数，具体这里应该有哪些参数，是由不同的方法自己定义的。

12.2 Item 支持的 Zabbix API 方法

在上一节中，了解了 Zabbix 向 API 推送数据的格式，在这一节中，主要学习如何找到需要的方法，以及返回值的含义。

在 Zabbix 官方文档里，有详细的 API 说明，Zabbix 2.2 的 API 文档在：<https://www.zabbix.com/documentation/2.2/manual/api>。我们具体分析一下 Item 的 API。

Item 的 API 文档结构如图 12-1 所示。

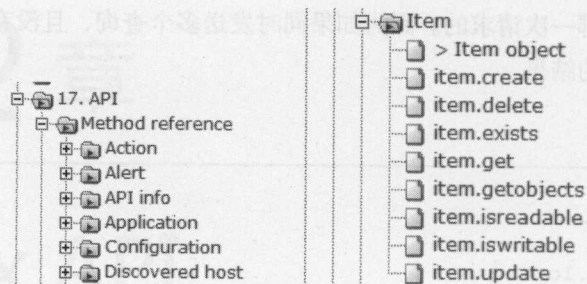


图12-1

其中，Action、Alert 等是 Zabbix 的资源。Item 资源下面有很多方法，其中 Item.object 相对特殊，它列出了在 API 中 Item 的所有属性，以及数字代表的含义。除了 Item.object，剩下几个就是 Item 支持的方法。

12.2.1 Item object

在 Item object（或者其他类似的 Event object、Trigger object）中，记录的是 Item 资源的所有属性。从 Zabbix 获取到 Item 资源对象的时候，返回的结果如下：

```
{
  jsonrpc: "2.0"
  -result: [
    -{
      itemid: "23287"
      type: "0"
      hostid: "10084"
      name: "Agent ping"
      key_: "agent.ping"
      delay: "60"
      history: "7"
      trends: "365"
      status: "0"
      value_type: "3"
      description: "The agent always returns 1 for this item. It could be used
in combination with nodata() for availability check."
```

```

}
]
id: 1
}

```

笔者已经删掉了一些属性了。这里可以看到很多返回值，都是一个数字，比如“value_type”，返回值是“3”，那单从返回值是不知道“3”是什么意思的，这时候就需要查 Item object 的文档了，如下。

value_type (required)	integer	Type of information of the item. Possible values : 0 – numeric float; 1 – character ; 2 – log ; 3 – numeric unsigned ; 4 – text.
-----------------------	---------	--

从文档中就可以知道，“value_type”为“3”的含义是这个 item 的“value_type”是“numeric unsigned”。

除了查询数字和具体含义的对应关系，Item object 还告诉我们，在创建一个 Item 的时候哪些字段是必须的。如果某个属性下面有“required”，那么就说明这个参数是必须的，比如下面表示监控时间间隔的“delay”。

delay (required)	integer	Update interval of the item in seconds.
------------------	---------	---

12.2.2 item.create

item.create 是新建 item 的方法，具体格式如下。

```

{
  "jsonrpc" : "2.0",
  "method" : "item.create",
  "params" : {
    "name" : "Free disk space on $1",
    "key_" : "vfs.fs.size[/home/joe/,free]",
    "hostid" : "30074",
    "type" : 0,
    "value_type" : 3,
    "interfaceid" : "30084",
    "applications" : [
      "609",

```

```

        "610"
    ],
    "delay" : 30
},
"auth" : "038e1d7b1735c6a5436ee9eae095879e",
"id" : 1
}

```

主要的参数在 `params` 中，参数列表可以参照 `Item object` 中的属性。

12.2.3 item.delete

`item.delete` 是删除 `item` 的方法，支持针对一个 `itemid` 或者多个 `itemid` 进行删除。一般要使用 `item.get` 方法来获取到需要删除的 `itemid` 后，调用 `item.delete` 进行删除。具体参数如下。

```

{
    "jsonrpc" : "2.0",
    "method" : "item.delete",
    "params" : [
        "22982",
        "22986"
    ],
    "auth" : "3a57200802b24cda67c4e4010b50c065",
    "id" : 1
}

```

12.2.4 item.exists

`item.exists` 是用来判断 `item` 是否存在的方法，将需要限制的条件写在 `params` 中就行了，参数如下。

```

{
    "jsonrpc": "2.0",
    "method": "item.exists",
    "params": {

```



```

    "host": "Linux Server",
    "key_": "vm.memory.size[available]"
  },
  "auth": "3a57200802b24cda67c4e4010b50c065",
  "id": 1
}

```

12.2.5 item.get

item.get 是最常用的方法之一，它的作用是根据给定的条件从 Zabbix 中获取符合条件的 Item。它支持很多的条件，我们先从最基本的例子看起。

```

{
  "jsonrpc": "2.0",
  "method": "item.get",
  "params": {
    "hostids": "10084"
  },
  "auth": "c745dad1222a0b4eb505707df47d6d81",
  "id": 1
}

```

上面这个参数，是用来获取 hostid 为 10084 的所有 Items 的，结果如下。

```

{
  jsonrpc: "2.0"
  -result: [
    -{
      itemid: "23287"
      hostid: "10084"
      value_type: "3"
      lastclock: "1397105347"
      lastns: "522187000"
      lastvalue: "1"
      prevvalue: "1"
    }
  ]
}

```

```
...
}
```

在这个例子中,使用 hostids 进行搜索,除了 hostids,还支持参数有 groupids、templateids 等。这里需要大家注意的是,在文档中,参数被分成了三类,表格中用一条横杠分开。

第一类, itemids、hostids 是根据 Item 的属性进行搜索的。

第二类,是限定的一些条件,如参数 “selectTriggers” 表示 get 到的 Items 列表要加上与之相关的 Triggers 的列表。

selectTriggers	query	Return triggers that the item is used in in the triggers property. Supports count.
----------------	-------	---

由上面文档中的内容可知,参数 “selectTriggers” 的值的类型是 “query”, “query” 的值有下面几个:

- ◎ short
- ◎ refer
- ◎ extend
- ◎ count

下面看个例子,如果要从 API 查询一个 Item,并且要列出和这个 Item 相关的 Triggers,分别要使用 4 个参数来查询,具体结果如下。

(1) short

```
-triggers: [
- {
triggerid: "13491"
}
]
```

(2) refer

```
triggers: [
- {
triggerid: "13491"
state: "0"
```

```
value_flags: "0"
```

```
}
```

```
]

```

(3) extend

```
triggers: [
```

```
-{
```

```
triggerid: "13491"
```

```
expression: "{13233}=1"
```

```
description: "Zabbix agent on {HOST.NAME} is unreachable for 5 minutes"
```

```
url: ""
```

```
status: "0"
```

```
value: "1"
```

```
priority: "3"
```

```
lastchange: "1396166407"
```

```
comments: ""
```

```
error: ""
```

```
templateid: "0"
```

```
type: "0"
```

```
state: "0"
```

```
flags: "0"
```

```
value_flags: "0"
```

```
}
```

```
]

```

(4) count

```
triggers: "2"
```

通过这个例子，大家应该能知道这 4 个参数的用途了吧。在其他类型为 query 的地方，也是如此使用。

第三类参数的作用，是将返回的数据进行过滤，比如前面举的例子：

```
{
```

```
"jsonrpc": "2.0",
```

```
"method": "item.get",
```

•213•

```
"params": {
    "hostids": "10084"
},
"auth": "c745dad1222a0b4eb505707df47d6d81",
"id": 1
}
```

可以加上 search 参数，即变成：

```
{
    "jsonrpc": "2.0",
    "method": "item.get",
    "params": {
        "hostids": "10084",
        "search": {
            "key_" : "system"
        }
    },
    "auth": "c745dad1222a0b4eb505707df47d6d81",
    "id": 1
}
```

这样就相当于在返回结果中过滤出“key_”中包含“system”字段的 Item 了，匹配方式类似于“%system%”。第三类的参数有很多，比如返回结果按照某一列排序的“sortfield”等，具体的列表可以查看官方文档。

12.2.6 item.getobjects

这个方法和 item.get 很类似，但它更加简单，写出需要满足的属性条件，则返回 item。格式如下。

```
{
    "jsonrpc": "2.0",
    "method": "item.getobjects",
    "params": {
```



```

    "host": "Zabbix server"
  },
  "auth": "3a57200802b24cda67c4e4010b50c065",
  "id": 1
}

```

12.2.7 item.isreadable/item.iswritable

这两个方法类似，用于判断登录的用户对某些 itemid 对应的 Item 是否有读或者写的权限，参数如下。

```

{
  "jsonrpc": "2.0",
  "method": "item.isreadable",
  "params": [
    "23298",
    "23323"
  ],
  "auth": "038e1d7b1735c6a5436ee9eae095879e",
  "id": 1
}

```

当“method”为“item.iswritable”时，判断是否有写权限。当“params”有多个 itemid 时，针对每一个 itemid 的结果，做“与”操作，即只要有一个 itemid 没有读权限，那么返回的就是 False。

12.2.8 item.update

item.update 是根据 itemid 更新 Item 的方法，它可以一次更新多个。参数如下。

```

{
  "jsonrpc": "2.0",
  "method": "item.update",
  "params": [
    {"itemid": "23975", "name": "frank"},

```

```
{ "itemid": "23847", "name": "frank" }
],
"auth": "c745dad1222a0b4eb505707df47d6d81",
"id": 1
}
```

在更新某个 Item 的时候，只要在参数中写上“itemid”即可，除了“itemid”以外，任何参数里的属性都会更新到对应的 itemid 上。

12.3 如何阅读Zabbix API文档

前面讲解了 Item 在 Zabbix API 中的操作，但 Zabbix API 支持的资源太多了，因篇幅限制，不可能讲得面面俱到。相比一个一个解释，笔者相信介绍如何去看 Zabbix API 文档是更加重要的。

比如现在要使用 item.get 方法，却不知道如何使用。首先看 item.get 的文档，发现它支持很多参数，但又不知道参数的值是哪种类型，文档里有“query”、“string”和“flag”，但不知道该怎么写。笔者根据官方文档，总结了 API 中涉及到的类型。

- bool：布尔值，值必须是“true”或者“false”。
- flag：只要值为 null 就认为是“true”，否则就是“false”。
- integer：整数。
- float：浮点数。
- string：字符串。
- text：长文本。
- timestamp：UNIX 时间戳。
- array：数组。
- object：一组数组。
- query：有 short, refer, extend, count 几种。其中 count 不是每一个“query”属性都可以使用的，具体看属性的说明。

有了这些，再看看文档中的例子，基本使用 API 就没有问题了。

Zabbix API 本身的功能和在页面上单击是一样的，API 的强大之处是可以让 Zabbix 脱离前端界面，以命令行的方式和人或者程序交互。这让 Zabbix 能够和其他系统一起组成整个运维自动化体系。

第 13 章

Zabbix分布式监控

Zabbix 是一个分布式的监控软件。在读者学习了 Zabbix 的各个功能模块之后，本章将介绍 Zabbix 分布式监控相关的内容，内容包括基本概念和部署。

13.1 两种分布式架构对比

Zabbix 支持两种分布式架构：一种是 Proxy 模式，一种是 Node 模式。Proxy 作为 Zabbix Server 的代理去监控服务器，并把数据汇总到 Zabbix Server。而 Node 本身就是一个完整的 Zabbix Server，使用 Node 可以将多个 Zabbix Server 组成一个有继承关系的分布式架构，两者的区别如表 13-1 所示。

表13-1

	Proxy	Node
轻量级	是	否
GUI 前端	否	是
是否可以独立运行	是	否
容易运维	是	否
本地 Admin 管理	否	是
中心化配置	是	否
产生通知	否	是

通过比较可以更好地理解二者的异同：Proxy 是个非常轻量级的组件，它的工作就是根据 Zabbix Server 的配置，从服务器收集一些监控数据，然后将监控数据发送给 Zabbix Server。Proxy 本身并不做任何对于监控数据的检查，比如检查这个数据是否需要报警等的逻辑。它所有的 Zabbix Server 操作都是在 Zabbix Server 端完成的。而对于 Node 模式来说，Node 本身就是一个具有完整功能的 Zabbix Server，它可以自行维护报警逻辑和监控数据，只是会把这些信息同步给它的父节点，形成一种形如继承关系的架构。

13.2 Proxy单级分布式架构

Proxy 架构是 Zabbix 中最简单的分布式架构，它的特点是部署简单，笔者在 PPTV 能够在 10 分钟以内就部署好一台 Proxy 并付诸使用。Proxy 的功能在前面已经说得比较清楚了，下面说说 Proxy 的工作原理。

Proxy 和 Zabbix Server 一样，它会从监控的 Host 获取监控数据，并保存在本地数据库。在满足一定条件后，会将这些数据发送给 Zabbix Server，再进行后续的操作，比如检查是否处罚 Trigger 之类。Proxy 相当于一个数据收集器，没有其他任何功能，它不会计算是否触发 Trigger、生成 Event 和发送报警。Zabbix Server 会每隔一段时间将自己的配置同步给 Proxy，这样一切的配置都是在 Zabbix Server 上配置然后同步到各个 Proxy 的，从而保证了分布式架构下的配置一致性。Zabbix 是以数据库中的信息为准的，如果出现了配置冲突，很有可能会造成 Zabbix 出问题，可能是逻辑问题也有可能直接 Crash。

使用 Proxy 可以减轻 Zabbix Server 的压力并解决网络上的一些问题。

在 Zabbix 中，容易出现性能瓶颈的是数据库端的操作。使用 Proxy 以后，相当于一部分压力会由 Proxy 的数据库承担，并且 Proxy 发送数据给 Zabbix Server 也是分批进行的，能够大大缓解 Zabbix Server 的压力。

另外，Proxy 也能解决很多网络问题。比如我国的互联网公司一般有多个 IDC 机房，由于一些原因，机房之间的互联互通并不是那么好，又或者是防火墙的问题造成 Zabbix Server 和服务器的网络连通性不好。这时候可以在每一个 IDC 机房放一个 Proxy 来监控本机房的服务器，我们只要解决 Proxy 和 Zabbix Server 两台服务器之间的问题就可以了。这在我国的网络环境是非常适用的，如图 13-1 所示。

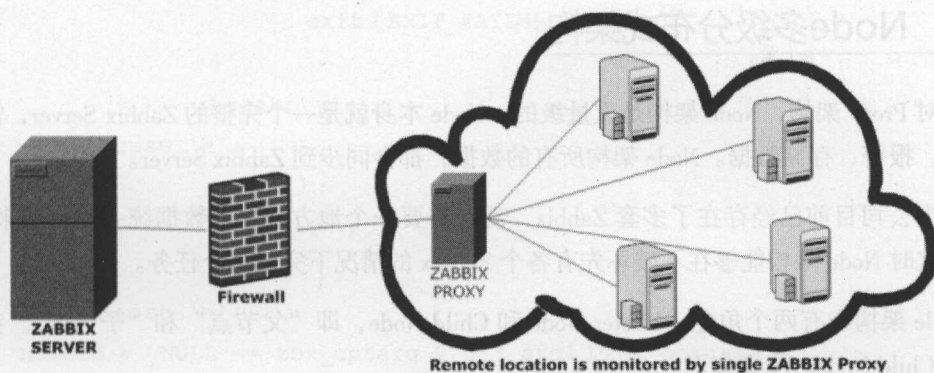


图 13-1

关于 Proxy 和 Zabbix Server 之间传输数据的可靠性,大家也可以放心,Proxy 会有本地缓存保证数据传输的完整性。在 PPTV,我们是电信一台 Proxy、网通一台 Proxy、多线机房一台 Proxy 的架构。

13.3 Proxy配置

Proxy 的配置非常简单,首先要有一台 Zabbix Server 和一台 Proxy,安装 Proxy 的方法和 Zabbix Server 类似,只是在 configure 编译的时候,参数有些调整。配置的前提是 Server 和 Proxy 都已经是在运行着的了。

进入“Administration”→“DM”,在右上角选择 Proxies,然后创建一个 Proxy,如图 13-2 所示。

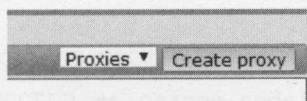


图 13-2

在配置 Proxy 的界面中,输入 Proxy 的 Hostname,选择哪些服务器归 Proxy 监控(后期可调整)。Proxy mode 可以选择 Active 或者是 Passive。Active 模式的 Proxy 会主动连接 Zabbix Server 和请求配置文件,而 Passive 模式的 Proxy 就是被动地等待 Zabbix 的连接。

大家要注意,Proxy 的 IP 也需要在 zabbix_agentd.conf 中的参数中配置,否则 Proxy 无法获取到 Agent 数据。Proxy 需要汇报给一个 Zabbix Server 而不能汇报给 Proxy,Proxy 不存在层级关系,只有从 Proxy 到 Zabbix Server 一种架构方式。

13.4 Node多级分布式架构

相对 Proxy 架构，Node 架构是重量级的。Node 本身就是一个完整的 Zabbix Server，需要收集数据、报警、存储数据。Node 架构所有的数据，都会同步到 Zabbix Server。

如果公司目前已经存在了多套 Zabbix，现在需要一个地方把这些数据统一集中起来管理，展示，这时 Node 架构能够在不破坏先有各个 Zabbix 的情况下完成这个任务。

Node 架构中有两个角色：Master Node 和 Child Node，即“父节点”和“子节点”。我们要先配置 Child Node。步骤如下。

(1) 每一个 Child Node 都有一个自己唯一的 NodeID，可以从 1 ~ 999。

(2) 关闭 Child Node 的 Zabbix server 时，最好同时也关闭运行其前端 PHP 的 Apache 或者 Nginx。

(3) 将数据库转变成 Node 架构使用的结构，运行以下命令：

```
zabbix_server -n <node id>
```

这里大家一定要注意，Zabbix 官方网站上特别强调了：这个命令只能跑一次，而且不能中途中断，否则可能会影响数据库。

我们来深入研究下，看看这个命令到底是干什么的。直接查看 zabbix_server 是看不到东西的，因为这是 C 编译出来的执行文件。查看 Zabbix 2.2.0 的源码，下面是 src/zabbix_server/server.c 的片段。

```
switch (ch)
{
    case 'c':
        CONFIG_FILE = zbx_strdup(CONFIG_FILE, zbx_optarg);
        break;
    case 'R':
        if (0 == strcmp(zbx_optarg, ZBX_CONFIG_CACHE_RELOAD))
            task = ZBX_TASK_CONFIG_CACHE_RELOAD;
        else
        {
            printf("invalid runtime control option: %s\n", zbx_optarg);
        }
    }
}
```

```

        exit(EXIT_FAILURE);
    }
    break;
case 'h':
    help();
    exit(-1);
    break;
case 'n':
    nodeid = (NULL == zbx_optarg ? 0 : atoi(zbx_optarg));
    task = ZBX_TASK_CHANGE_NODEID;
    break;
case 'V':
    version();
    exit(-1);
    break;
default:
    usage();
    exit(-1);
    break;
}

```

可以追踪到：

```

switch (task)
{
    case ZBX_TASK_CHANGE_NODEID:
        exit(SUCCESS == change_nodeid(nodeid) ? EXIT_SUCCESS :
EXIT_FAILURE);
        break;
    default:
        break;
}

```

接着寻找 `change_nodeid` 方法，发现了 `src/zabbix_server/utils/nodechange.c`，看其中的 `printf` 代码，即在屏幕中输出的内容，一共有下面这些步骤。

(1) Dropping foreign keys

(2) Converting tables

(3) Creating foreign keys

我们先看第一步把数据库的外键去掉的代码：

```
for (i = 0; NULL != db_schema_fkeys_drop[i]; i++)
{
    DBexecute ("%s", db_schema_fkeys_drop[i]);
    printf (".");
    fflush (stdout);
}
```

SQL 语句是在 db_scheme_fkeys_drop 中，使用 grep 来搜索：

```
$ grep -r 'db_schema_fkeys_drop' *
libs/zbxdbhigh/dbschema.c:const char      *const db_schema_fkeys_drop[] = {
libs/zbxdbhigh/dbschema.c:const char      *const db_schema_fkeys_drop[] = {
libs/zbxdbhigh/dbschema.c:const char      *const db_schema_fkeys_drop[] = {
libs/zbxdbhigh/dbschema.c:const char      *const db_schema_fkeys_drop[] = {
libs/zbxdbhigh/dbschema.c:const char      *const db_schema_fkeys_drop[] = {
zabbix_server/utils/nodechange.c:          for (i = 0; NULL != db_schema_
fkeys_drop[i]; i++)
zabbix_server/utils/nodechange.c:      DBexecute ("%s", db_schema_fkeys_
drop[i]);
```

再到 dbschema.c 中，发现针对不同数据库有不同的 SQL 语句，比如下面是 MySQL 的：

```
const char      *const db_schema_fkeys_drop[] = {
    "ALTER TABLE `hosts` DROP FOREIGN KEY `c_hosts_1`",
    "ALTER TABLE `hosts` DROP FOREIGN KEY `c_hosts_2`",
    "ALTER TABLE `hosts` DROP FOREIGN KEY `c_hosts_3`",
    "ALTER TABLE `group_prototype` DROP FOREIGN KEY `c_group_
prototype_1`",
    "ALTER TABLE `group_prototype` DROP FOREIGN KEY `c_group_
prototype_2`",
```

```

"ALTER TABLE `group_prototype` DROP FOREIGN KEY `c_group_
prototype_3`",
"ALTER TABLE `group_discovery` DROP FOREIGN KEY `c_group_
discovery_1`",
"ALTER TABLE `group_discovery` DROP FOREIGN KEY `c_group_
discovery_2`",
...
}

```

接着看 Converting talbes 的步骤，下面是核心代码：

```

if (0 == strcmp (tables[i].fields[j].name, tables[i].recid))
{
    prefix = ZBX_DM_MAX_HISTORY_IDS * (zbx_uint64_t) nodeid;
    if (0 != (tables[i].flags & ZBX_SYNC))
        prefix += ZBX_DM_MAX_CONFIG_IDS * (zbx_uint64_t) nodeid;
}

/* relations */
else if (NULL != tables[i].fields[j].fk_table)
{
    r_table = DBget_table (tables[i].fields[j].fk_table);
    assert (NULL != r_table);
    prefix = ZBX_DM_MAX_HISTORY_IDS * (zbx_uint64_t) nodeid;
    if (0 != (r_table->flags & ZBX_SYNC))
        prefix += ZBX_DM_MAX_CONFIG_IDS * (zbx_uint64_t) nodeid;
}

/* special processing for table 'profiles' */
else if (0 == strcmp ("profiles", tables[i].table))
{
    convert_profiles (nodeid, tables[i].fields[j].name);
    continue;
}
else
    assert (0);

DBexecute ("update %s set %s=%s+" ZBX_FS_UI64 " where %s>0",

```

```

tables[i].table,
tables[i].fields[j].name,
tables[i].fields[j].name,
prefix,
tables[i].fields[j].name);

```

我们反过来看这段代码，最后的 DBexecute 显然是在执行一些 SQL，SQL 的内容是在增加一些值，即将 tables[i].fields[j].name 增加 prefix 这么多（ZBX_FS_UI64 是个占位符，我们理解为 %s 就行了）。tables 这个数组也是在 dbschema.c 中定义的，它的内容就是 Zabbix 数据库中的结构，tables[i].fields[j].name 可以理解为遍历数据库的每一张表。我们可以看到，这条 SQL 最关键的是 prefix。prefix 的赋值过程是这样的：

```

prefix = ZBX_DM_MAX_HISTORY_IDS * (zbx_uint64_t) nodeid;

```

ZBX_DM_MAX_HISTORY_IDS 是在 include/db.h 中定义的，它的值是 10000000000000000000：

```

#define ZBX_DM_MAX_HISTORY_IDS      (zbx_uint64_t) __UINT64_C
(10000000000000000000)

```

如果 tables 的 flags 字段为 1，即 “flags & ZBX_SYNC” 为 true 的时候，prefix 还要再加上如下内容：

```

prefix += ZBX_DM_MAX_CONFIG_IDS * (zbx_uint64_t) nodeid;
#define ZBX_DM_MAX_CONFIG_IDS      (zbx_uint64_t) __UINT64_C(1000000000000)

```

从代码可以看出，prefix 由这两个数字乘以 nodeid 得到，比如 nodeid 为 3，那么 prefix 就是 3000000000000000000。每一个 Item 在 Zabbix 数据库中都有一个自己的 id，比如 10001 这个 id，建立 Node 架构后，它的 itemid 就会变为 00000000010001。

下面还有一个对 prefix 的操作：

```

prefix += ZBX_DM_MAX_CONFIG_IDS * (zbx_uint64_t) nodeid;

```

这个操作只当某个数据库表的 flags 字段为 ZBX_SYNC 的时候才会进行，只不过 ZBX_DM_MAX_CONFIG_IDS 的值是 10000000000000000000。

我们来看看 flags 字段的值是什么，首先我们发现 talbes 数组，它的类型是 ZBX_TABLE。继续追溯，可以从 dbschema.h 中找到 ZBX_TABLE 的结构定义。再进一步，可以知道上面这一行是一个 ZBX_FIELD，根据它的定义，我们就知道 flag 字段原来是倒数第二个字段，随便在

dbschema.c 中找一行表示结构的描述：

```
 {"ipmi_available", "0", NULL, NULL, 0, ZBX_TYPE_INT, ZBX_NOTNULL | ZBX_SYNC, 0},
```

发现它的 flag 值是 ZBX_NOTNULL | ZBX_SYNC。这种形式是非常典型的 C 语言中表示属性的用法，ZBX_NOTNULL | ZBX_SYNC 表示这一行有两个属性，分别是 ZBX_NOTNULL 和 ZBX_SYNC。再结合代码逻辑，下面这个逻辑与的含义就是这个数据库表是具有 ZBX_SYNC 属性的。

```
 flags & ZBX_SYNC
```

后面的逻辑就简单了，就是如果数据库表有 ZBX_SYNC 属性，那么 prefix 还要再增加 ZBX_DM_MAX_CONFIG_IDS*nodeid 这么多。具有 ZBX_SYNC 这个属性，就表示这个数据库在分布式架构中，是需要在不同节点间传输的。

搞清楚了 prefix 的来龙去脉，那为什么要将表中某个字段加上 prefix 呢？这里就要简单说一下 Zabbix 数据库表结构了，对于任何一个资源（Item、Host 等），在表中都有一个自己的 id，比如 itemid、hostid 等，这个 id 是该表的主键。在分布式架构中，所有子节点的数据要同步到父节点中，难免会出现子节点中 itemid（或其他 id）相同的情况，所以需要将不同子节点的 id 区分开，要对 id 加上一个前缀。比如 10001 这个 itemid，再加上 prefix 后，可能就会变成 300000000010001。

最后一步就是创建新的外键了，和删除外键类似，从 db_schema_fkeys 中获取 SQL，形如：

```
const char      *const db_schema_fkeys[] = {
    "ALTER TABLE `hosts` ADD CONSTRAINT `c_hosts_1` FOREIGN KEY (`proxy_`
hostid`) REFERENCES `hosts` (`hostid`)",
    "ALTER TABLE `hosts` ADD CONSTRAINT `c_hosts_2` FOREIGN KEY
(`maintenanceid`) REFERENCES `maintenances` (`maintenanceid`)",
    "ALTER TABLE `hosts` ADD CONSTRAINT `c_hosts_3` FOREIGN KEY
(`templateid`) REFERENCES `hosts` (`hostid`) ON DELETE CASCADE",
    "ALTER TABLE `group_prototype` ADD CONSTRAINT `c_group_prototype_1`
FOREIGN KEY (`hostid`) REFERENCES `hosts` (`hostid`) ON DELETE CASCADE",
    "ALTER TABLE `group_prototype` ADD CONSTRAINT `c_group_prototype_2`
FOREIGN KEY (`groupid`) REFERENCES `groups` (`groupid`)",
    "ALTER TABLE `group_prototype` ADD CONSTRAINT `c_group_prototype_3`
FOREIGN KEY (`templateid`) REFERENCES `group_prototype` (`group_prototypeid`)
ON DELETE CASCADE",
```

```
...
```

```
}
```

现在大家了解了整个创建 Node 的过程，应该知道为什么命令只能跑一次了吧。因为如果运行多次，在 update 数据库的时候，prefix 会被加很多次，从而造成数据库错乱。而运行到一半就退出，虽然说可以修复，但是会非常困难。我们需要知道之前运行到什么地方，然后还要将 Zabbix 的这些逻辑反推回去创建修复的 SQL。

建议各位在操作的时候，先备份一下数据库，以防万一。

接下来我们看看在设置完 Node 后的操作。

先在 Master 节点上进行设置，从“Administration”→“DM”中进入，在右上角的下拉框中选择“Nodes”选项。如果单击进去后发现显示的是“Your setup is not configured for distributed monitoring”，那就是前面几步没有设置好，需要再检查一下，一个是 zabbix_server.conf 中的 NodeID 参数，一个是否设置了数据库。

首先要设置架构中的 Master Node，即父节点，单击“Local Node”按钮可以看到当前节点的装填。单击“New node”按钮创建一个子节点，在呈现的设置界面中，需要设置得都比较简单，自行设置即可。

接下来看 Child Node（子节点）的设置。同样的，从“Administration”→“DM”中进入，单击“New node”按钮，根据提示设置需要的属性，需要注意的是 ID 要设置的和这个 Child node 的 zabbix_server.conf 中的 NodeID 一样。Type 设置为 Child。

一切设置完以后，启动 Master Node 的 zabbix_server 就可以了。

最后看如何使用 Node 架构。当设置完整套 Node 架构后，在前端的右上角，有一个下拉列表“Current Node”，在其中可以选择要查看的节点的数据。

第 14 章

Zabbix系统优化

Zabbix 虽然是一个优秀的开源软件，但是所有的开源软件都有“坑”，Zabbix 也不例外。它的一个大“坑”就是性能。当 Zabbix 规模增长以后，Zabbix 的性能会变得糟糕，特别是对于不熟悉 Zabbix 的人来说。

总的来说，Zabbix 变慢的原因有如下 4 个。

- 数据量太大了，vps 太高，Zabbix 来不及处理。
- Housekeeper 设置不当，数据库体积变大。
- 前端用户太多，查询过多的数据。
- Triggers 太复杂。

要做调优，必须有数据来支撑，Zabbix 本身提供了很多监控项来监控 Zabbix 本身性能数据，大家一定要合理运用起来。

14.1 Zabbix内部运行机制

“知己知彼，百战百胜”，在调优 Zabbix 之前，先看看 Zabbix 内部的运行机制图（图 14-1），可以理解一下 Zabbix Server 启动的各种各样的进程的作用，以及工作的流程。

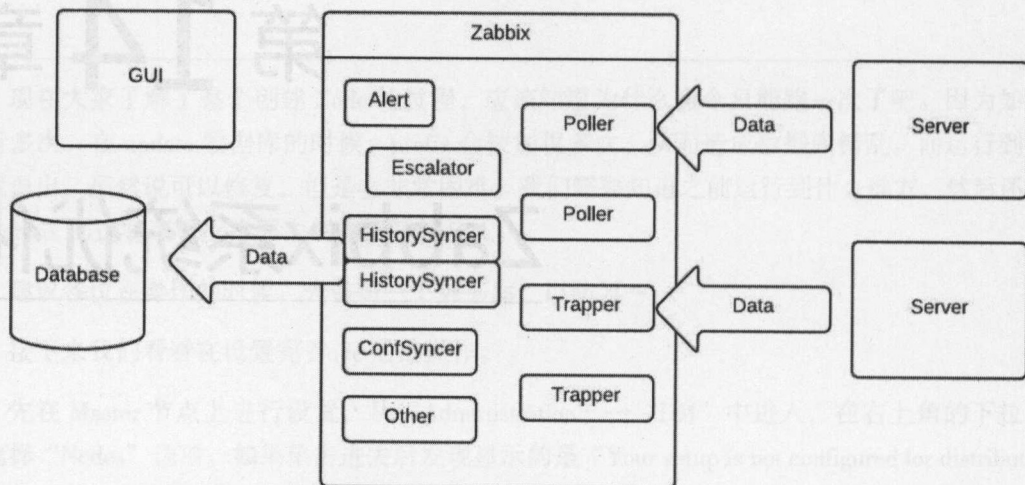


图14-1

简单地说，Poller 和 Trapper 抓取或者接受数据，Syncer 将数据写入数据库。所以，当出现问题的时候，可以先看看是否是这些进程的数量变少了。

14.2 Items过多造成性能下降

有很多朋友问过我：“Zabbix 性能差怎么办？”而且，我发现他们的 Zabbix 中 Item 的 interval 调整得非常小，都在几秒左右，从而 VPS（value per second，每秒数据量）非常高。我们先来看 PPTV 的一个 interval 分布：

```
SQL>select delay, count(*), round(count(*)/(select count(*) from
items where status=0)*100,2)||' %' as percent from items where status=0
group by delay order by 2 desc;
```

DELAY	COUNT (*)	PERCENT
600	119489	18.04%
86400	103224	15.59%
300	90168	13.62%
2400	79051	11.94%
7200	62286	9.41%

1200	53981	8.15%
120	37741	5.7%
240	34252	5.17%
3600	25578	3.86%
60	19412	2.93%
51840	15251	2.3%
180	8026	1.21%
172800	4125	.62%
900	3344	.5%
120960	3309	.5%
30	2942	.44%
150	16	0%
6000	5	0%
1800	4	0%

在这里看一下 VPS 是如何计算的, 就知道 Item 的 interval 对于 Zabbix 系统的影响有多大。首先找到 Reports-Status of Zabbix, 如图 14-2 所示。

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (monitored/not monitored/templates)	52	11 / 0 / 41
Number of items (monitored/disabled/not supported)	121	113 / 0 / 8
Number of triggers (enabled/disabled) [problem/ok]	65	65 / 0 [1 / 64]
Number of users (online)	2	1
Required server performance, new values per second	1.98	-

图14-2

进入后, 发现 PHP 文件叫做 report1.php, 打开后发现了以下代码:

```
$reportWidget->addItem(make_status_of_zbx());
```

然后去找 make_status_of_zbx 这个方法:

```
$ grep -r 'make_status_of_zbx' *
dashboard.php:                                $stszbx = make_status_of_zbx();
include/blocks.inc.php:function make_status_of_zbx() {
report1.php:$reportWidget->addItem(make_status_of_zbx());
```

可以看到在 dashboard.php、blocks.inc.php 和 report1.php 这三个文件中出现了 make_status_of_zbx，其中 report1.php 可以排除，而 dashboard.php 则是前端另一个显示 VPS 的地方，所以对于这个方法的设置，一定是在 include/blocks.inc.php 中。打开这个文件找到生成 VPS 的地方：

```
$table->addRow(array(_('Required server performance, new values per second'), $status['qps_total'], ' - '))
```

可以发现，VPS 这个数据是从 \$status 中获取的，\$status 是怎么获取的，找到如下代码：

```
$status = get_status();
```

继续寻找 get_status 方法的定义：

```
$ grep -r 'get_status' *
include/blocks.inc.php: show_messages(); // because in function get_status(); function clear_messages() is called when fsockopen() fails.
include/blocks.inc.php: $status = get_status();
include/classes/class.cserverinfo.php: $status = get_status();
include/func.inc.php: function get_status() {
```

发现了对于 get_status 的定义是在 include/func.inc.php 中，找到对 qps 的定义：

```
$row = DBfetch(DBselect(
    'SELECT SUM(1.0/i.delay) AS qps'.
    ' FROM items i,hosts h'.
    ' WHERE i.status='.ITEM_STATUS_ACTIVE.
    ' AND i.hostid=h.hostid'.
    ' AND h.status='.HOST_STATUS_MONITORED.
    ' AND i.delay<>0'
));
$status['qps_total'] = round($row['qps'], 2);
```

其中最关键的就是 SUM(1.0/i.delay)，delay 就是 Item 每两次从 Agent 获取数据的时间间隔。这时候我们清楚了，VPS 这个数值，反映了 delay 的分布情况。

找到问题了，应该怎么解决呢？方法就是调整 Items 的监控间隔，这是非常简单有效的。如果要精确一些，可以通过看 Zabbix 对于本身队列的监控，来确定是什么 Item 造成慢的。比如从图 14-3 所示的情况中，就能看出是 SNMPv1 agent 造成的问题。

QUEUE OF ITEMS TO BE UPDATED						
Items	5 seconds	10 seconds	30 seconds	1 minute	5 minutes	More than 10 minutes
Zabbix agent	6	2	3	0	0	0
Zabbix agent (active)	0	0	0	0	0	0
Simple check	1	0	0	0	0	0
SNMPv1 agent	0	0	0	0	0	1
SNMPv2 agent	0	0	0	0	0	0
SNMPv3 agent	0	0	0	0	0	0
Zabbix internal	0	0	0	0	0	0
Zabbix aggregate	0	0	0	0	0	0
External check	0	0	0	0	0	0
Database monitor	0	0	0	0	0	0
IPMI agent	0	0	0	0	0	0
SSH agent	0	0	0	0	0	0
TELNET agent	0	0	0	0	0	0
Calculated	0	0	0	0	0	0

图 14-3

但很多情况下，不是由某一类 Item 引起的，所以，全部从数据库更新监控的间隔，是一个行之有效的方法。

如果上面的方法都没有效果了，那就只有一个办法了——拆分架构，使用分布式架构。这样会减轻 Zabbix Server 的压力，将这些压力分摊到 Proxy 上去，如图 14-4 所示。

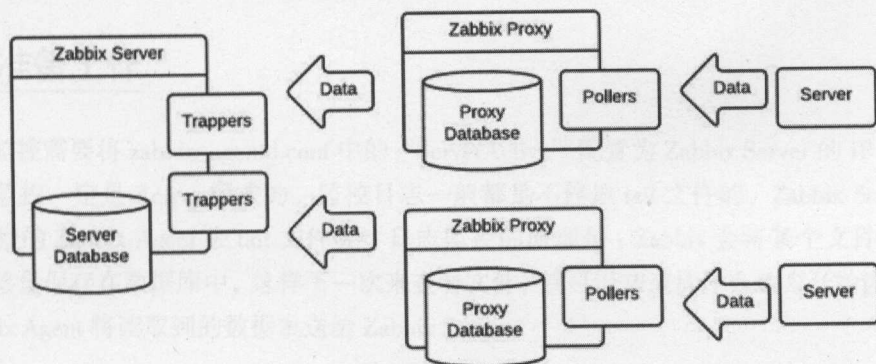


图 14-4

在这种分布式架构中，性能瓶颈可能会出现在数据库中。

14.3 数据库及其他调优

对于数据库调优，其实非常简单，history 表每天、每周或每月分区，这个时间间隔要看具体情况，然后关闭 housekeeper 就行了。这个办法能解决 80% 的数据库问题。从 Zabbix 的运行机制可以知道，history 表的数据非常大，如果开启 housekeeper，那么在 housekeeper 执行的过程中，Zabbix 会响应很慢，因此推荐将 history 分区，关闭 housekeeper。

如果 Zabbix 没有问题，但数据库的数据进入比较慢，可以尝试调高 zabbix_server.conf 中的 StartDBSyncers，这个参数是设置将数据从 Zabbix 写入数据库的进程的多少的。

在 Zabbix 2.2 中，调优已经是个非常简单的问题了。因为在 Linux 中，使用“ps -ef”或者其他类似命令，就可以看到 Zabbix 各个进程都在忙些什么，以及它们的性能数据。比如下面这条：

```
./zabbix_server: history syncer #4 [synced 0 items in 0.000044 sec, idle 5 sec]
```

就能很清晰地看出 Zabbix 中负责 history 部分工作的进程的忙碌程度了。寻找到忙碌的进程，调大它在 zabbix_server.conf 中的进程数，就能有效地缓解问题。



第 15 章

轻量级日志监控应用

过去的一年，笔者一直在唯品会从事日志方面的工作，所以对日志的需求有一定的了解。非常常见的需求就是如果日志中出现了某个关键字，那么就要报警。Zabbix 本身自带了这样的功能，在这一章中会介绍如何使用这个功能。

场景是这样的：有一个系统，会往 `/tmp/app.log` 中打印日志，日志格式形如“2014-03-31 08:00:00 OK”。如果发现问题，日志中的“OK”会变为“ERROR”。如果出现了“ERROR”，我们要获取“ERROR”后面的出错信息，并且发邮件通知。

15.1 准备工作

日志监控需要将 `zabbix_agentd.conf` 中的“ServerActive”配置为 Zabbix Server 的 IP，因为日志类型的监控一定是 Active 模式的。监控日志一般都是不停地 tail 文件的，Zabbix Server 怎么能控制远程的 Zabbix Agent 去 tail 文件呢？日志监控的原理是：Zabbix 会将某个文件 tail 到哪里这个偏移量保存在数据库中，这样下一次来查看文件，就可以知道从什么地方开始读日志了。然后 Zabbix Agent 将读取到的数据发送给 Zabbix Server。

除了 Zabbix 前端需要的配置外，还要设置好 Zabbix Agent 中 `zabbix_agentd.conf` 里的 `ServerActive`，另外，要确认 Host name 和 Zabbix Server 中配置得一样，是 Active 模式所需要的，如果两边配置不正确，那么向 Zabbix Server 发送数据时，Server 就不知道到底是算在哪个 Host 上的了。

15.2 添加 Item

接下来看看添加 Item 的配置。需要配置的有以下几项。

(1) Type：一定要选择“Zabbix agent (active)”，否则 Item 会认为是“ZBX_UNSUPPORTED”。

(2) Type of information：选择 Log。

(3) Log time format：根据我们的例子设置，“yyyy-MM-dd hh:mm:ss”，对于这个参数笔者也曾一度不清楚是干什么的，在下面会有解释。

配置中的“Update interval”设置为 5 秒，为的是能尽快看到监控的效果。在全部设置完后保存，就可以了，如图 15-1 所示。

The screenshot shows the Zabbix Item configuration interface. The 'Name' field is 'testLog'. The 'Type' dropdown is set to 'Zabbix agent (active)'. The 'Key' field is 'log[/tmp/app.log,ERROR]' with a 'Select' button. The 'Type of information' dropdown is set to 'Log'. The 'Update interval (in sec)' is '5'. The 'History storage period (in days)' is '90'. The 'Log time format' is 'yyyy-MM-dd hh:mm:ss'. The 'New application' field is empty. The 'Applications' list includes '-None-', 'CPU', 'Filesystems', 'General', 'log' (which is highlighted), and 'Memory'. The 'Description' field is an empty text area. At the bottom, the 'Enabled' checkbox is checked.

图15-1

15.3 测试

可以在 Zabbix Agent 所在的服务器进行一些测试：

```
echo "ERROR" >> /tmp/app.log
```


同时可以观察 zabbix_agentd.log, 会有类似下面的日志 :

```
13586:20140331:233049.321 JSON before sending [{
  "request": "agent data",
  "data": [
    {
      "host": "Zabbix server",
      "key": "log[/tmp/app.log,ERROR]",
      "value": "2014-03-31 23:30:40 ERROR",
      "lastlogsize": 52804,
      "clock": 1396279844,
      "ns": 310724000},
    {
      "host": "Zabbix server",
      "key": "log[/tmp/app.log,ERROR]",
      "value": "2014-03-31 23:30:41 ERROR",
      "lastlogsize": 52830,
      "clock": 1396279844,
      "ns": 311017000},
    {
      "host": "Zabbix server",
      "key": "log[/tmp/app.log,ERROR]",
      "value": "2014-03-31 23:30:41 ERROR",
      "lastlogsize": 52856,
      "clock": 1396279844,
      "ns": 311296000}],
    "clock": 1396279849,
    "ns": 321740000}]
13586:20140331:233049.322 JSON back [{
  "response": "success",
  "info": "processed: 3; failed: 0; total: 3; seconds spent: 0.000368"}]
```

可以发现, 一共往日志里输出了三次 “ERROR”, 所以向 Zabbix Server 发送了三次请求。再来看看前端界面, 如图 15-2 所示。

Timestamp	Local time	Value
2014.Mar.31 23:30:44	-	2014-03-31 23:30:41 ERROR
2014.Mar.31 23:30:44	-	2014-03-31 23:30:41 ERROR
2014.Mar.31 23:30:44	-	2014-03-31 23:30:40 ERROR
2014.Mar.31 23:30:44	-	2014-03-31 23:30:40 ERROR
2014.Mar.31 23:29:59	-	2014-03-31 23:29:56 ERROR
2014.Mar.31 22:33:07	2014.Mar.31 22:33:04	2014-03-31 22:33:04 ERROR
2014.Mar.31 22:33:07	2014.Mar.31 22:33:02	2014-03-31 22:33:02 ERROR
2014.Mar.31 21:55:11	2014.Mar.31 21:55:09	2014-03-31 21:55:09 ERROR

图15-2

为什么有几个数据的“Local time”为空呢？因为笔者开始时不明白设置 Item 时候的“Log time format”是干什么用的，所以将它设置为空，想看看有什么变化，结果就是“Local time”变成空了。所以，Zabbix 会根据监控日志时配置的“Log time format”来解析这条日志被打印出来的时间。大家可以仔细观察一下，“Local time”不为空的那几行里的时间，跟日志里的时间是吻合的。

另外，可以看到，使用“log”这个 key 来监控日志，只有当日志中出现设置的关键字的时候，Zabbix Agent 才会将出现关键字的日志发送到 Zabbix Server。确实如此，这个监控的设计初衷就是这样，只有当出错的时候，才将出错的日志发送过来。

15.4 配置报警

报警的配置非常简单，在 Host（或者 Template）上找到 Items，单击 wizard 里的图标，在弹出的菜单中选择“Create trigger”，如图 15-3 所示。

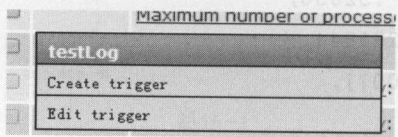


图15-3

弹出的界面和平时设置 Trigger 的界面不太一样，这也是我为什么强烈建议在这里配置日志文件监控的 Trigger 的原因。大家要知道，在以前 Trigger 中配置字符串匹配的语法是非常复杂和麻烦的，99% 的人第一次配置都会出错。

现在就很简单了，选择一个监控日志的 Item，在 Expression 中填写关键字，“iregexp”表示“是否忽略大小写”，“Include”表示日志要包含填写在 Expression 中的关键字，“Exclude”则相反。

使用“AND”和“OR”可以组合多个条件，比如出现“ERROR”但没有出现“Exception”等。每设置好一个，单击“Add”按钮添加。图 15-4 是笔者配置的 Trigger 的一个示例，它的作用是如果出现“ERROR”，那么 Trigger 就是 PROBLEM 状态。

The screenshot shows the Zabbix Trigger configuration interface. The 'Description' field contains 'app.log is error'. The 'Item' field contains 'test1.log'. The 'Expression' section shows a configuration with 'regex(ERROR)' and 'Include' type. The 'Severity' is set to 'Not classified'. The 'Action' section shows a single action 'Delete'. The 'Comments' and 'URL' fields are empty. The 'Disabled' checkbox is unchecked. The 'Save' and 'Cancel' buttons are at the bottom right.

图 15-4

再 echo 几个“ERROR”到日志文件里，就能看到 Trigger 变成 PROBLEM 状态了，如图 15-5 所示。

Time	Description	Status
31 Mar 2014 23:58:34	app.log is error	PROBLEM
31 Mar 2014 23:58:34	app.log is error	PROBLEM
31 Mar 2014 23:58:34	app.log is error	PROBLEM

图 15-5

后续配置 Action 的操作，这里不做赘述。

15.5 轮转的日志文件

日志轮转是经常碰到的问题，如果要监控的日志文件每天都会轮转，应该如何处理呢？因为我们在“log[/tmp/app.log,ERROR]”中配置的是 /tmp/app.log，而日志文件每天轮转的文件名是不同的。在这里，Zabbix 支持使用 logrt 这个 Item 来监控。“logrt”中的“rt”的意思是“rotate”，即常说的“轮转”。

logrt 的使用也非常简单，第一个参数使用能够匹配到日志文件名的正则表达式就行了。比

如 /tmp/app.log 会生成 /tmp/app.log.1 和 /tmp/app.log.2，我们就可以这样配置 logrt：

```
logrt[/tmp/app.log.*,ERROR]
```

需要注意的是，匹配日志文件的正则表达式，只支持对于文件名的正则匹配，对于目录是不支持的。

15.6 获取关键字

有的时候我们不要一条完整的报错信息，而是只需要出错的关键字，这时可以使用 Zabbix 2.2 提供的新特性——支持从日志中匹配出我们需要的信息。之前定义它只会返回“ERROR”，现在在“ERROR”后面会加上一些详细的出错信息，比如“ERROR database cannot connet”。我们看看需要怎样配置。

需要的配置非常简单，一个是用正则把需要的信息过滤出来，另外就是选择匹配出的是第几个组。key 配置为“log[/tmp/app.log,ERROR(.*)\$,,,\\1]”，在 ERROR 后面增加了一些正则表达式，表示匹配“ERROR”后的所有内容直至这一行的末尾，中间几个逗号是因为省略了不需要的几个参数，最后的“\\1”表示抓取的是匹配出的第一分组。

向 /tmp/app.log 中输出了“2014-04-01 08:20:31 ERROR hahaha”，能看到已经匹配出来了，“Local time”为空是因为没有设置“Log time format”，如图 15-6 所示。

Timestamp	Local time	Value
2014.Apr.01 08:20:34 -		hahaha
2014.Apr.01 08:20:34 -		hahaha
2014.Apr.01 08:20:34 -		hahaha
2014.Apr.01 08:20:34 -		hahaha
2014.Apr.01 08:20:34 -		hahaha
2014.Apr.01 08:20:34 -		hahaha

图15-6

第 16 章

Zabbix 数据库表结构解析



第四部分 设计篇

- ★ 第 16 章 Zabbix 数据库表结构解析
- ★ 第 17 章 History 和 Trends
- ★ 第 18 章 Zabbix 和数据库交互详解
- ★ 第 19 章 Zabbix 2.2 新功能介绍
- ★ 第 20 章 Zabbix 内置监控项实现

第 16 章

Zabbix数据库表结构解析

由于 Zabbix 前端效率极低，大批量更新会造成 Oracle tx 锁，所以对于这种大批量的更新，一般都用数据库语句直接 update。通过 patch PHP 代码的方法可以解决这个问题（Zabbix 本身的类似问题就是这样解决的），但是 PHP 可能有的人会不太擅长，而且太长。数据 update 的速度会快很多。这时需要对 Zabbix 数据库的数据结构有清晰的了解。

另外，Zabbix 收集了大量的裸数据，其他人可以通过这些数据来进行分析，同样，也需要了解数据库的结构。

在笔者使用过程中，也是摸石头过河，一边摸索一边使用。这里会对 Zabbix 数据库的表结构和常用的操作做一些说明。

注意：

- ◎ Zabbix 数据库中的表的名称都是复数，比如存放 Host 信息的表的名字是 Hosts 等。
- ◎ 数据库操作有风险，一旦出问题会造成 Zabbix crash。需要谨慎操作。
- ◎ 普通的查询可以在备库上进行。两边数据是实时同步的。

16.1 表结构概述

Zabbix 的数据库设计是很有特点的，针对 Zabbix 中的每一个资源，都有一张表与其对应，比如 hosts 表、items 表等。而每一张表中，都有一个 id 字段，比如 hosts 表中有 hostid 列，

items 表中有 itemid 列。而资源之间的关联关系，是通过外键来完成的。比如 Host 和 Item 的关联关系，就是在 items 表中使用 hostid 与 hosts 表中的资源进行关联的。

下面以 hostid 为例进行说明，其他 itemid、trigger 等都与之类似。

hostid 是每一个 Host 的唯一标识，从数据库查询的时候，一般都是以 hostid 为查询条件。点开一个 Host，然后看 URL，例如：

```
http://192.168.201.234/zabbix/hosts.php?form=update&hostid=10108&groupid=0&sid=27716d11b8954723
```

可以看到 URL 里的 GET 参数有以下几个。

- ◎ form：表示当前页面的操作，这里的 update 是因为笔者是从“Configuration”→“Hosts”中单击 host 进入的，所以是一个更新的操作。
- ◎ hostid：单击的 Host 的 hostid。
- ◎ groupid：这里不需要 groupid 这个字段，所以 0 没有意义。
- ◎ sid：sessionid，标识用户。

这里顺便说一句，Zabbix 的前端界面的 URL，会有很多像上面的 URL 那样。有的是 itemid，有的是 triggerid，我们更改一下这个 id，就自然能够跳转到对应的界面上去了。这一点正是 Zabbix 的灵活之处，在我们进行二次开发的时候，非常有用。

有的朋友要说了，我只要用好 Zabbix 就行了，不想去了解 Zabbix 的数据库表结构，觉得没必要也没用。其实非也，从 Zabbix 数据库的表结构，我们可以知道 Zabbix 资源的数据结构。另外，活用 SQL 查询 Zabbix 数据库，能够提升效率。比如想看某个机房的网卡出口流量之和，怎么办呢？我们可能会定义一个很复杂的 aggregate 类型的 Item，然后在前端点来点去。但是如果用 SQL，就一条 SQL 就可以非常简单地解决问题了。

另外，Zabbix 的水平高低，或者说是否真正了解 Zabbix 的一个非常大的标志，就是是否了解 Zabbix 的数据库。大家可能看这一节很吃力，但没关系，如果是刚接触 Zabbix 不久，这也是正常的，可以先看一遍留个印象，等之后对 Zabbix 有了些了解以后再来仔细学习。

16.2 Hosts表

“Host”就是指一台被监控的机器。Hosts 表结构如下。

```
mysql> desc hosts;
```

Field	Type	Null	Key	Default	Extra
hostid	bigint(20) unsigned	NO	PRI	NULL	
proxy_hostid	bigint(20) unsigned	YES	MUL	NULL	
host	varchar(64)	NO	MUL		
status	int(11)	NO	MUL	0	
disable_until	int(11)	NO		0	
error	varchar(128)	NO			
available	int(11)	NO		0	
errors_from	int(11)	NO		0	
lastaccess	int(11)	NO		0	
ipmi_authtype	int(11)	NO		0	
ipmi_privilege	int(11)	NO		2	
ipmi_username	varchar(16)	NO			
ipmi_password	varchar(20)	NO			
ipmi_disable_until	int(11)	NO		0	
ipmi_available	int(11)	NO		0	
snmp_disable_until	int(11)	NO		0	
snmp_available	int(11)	NO		0	
maintenanceid	bigint(20) unsigned	YES	MUL	NULL	
maintenance_status	int(11)	NO		0	
maintenance_type	int(11)	NO		0	
maintenance_from	int(11)	NO		0	
ipmi_errors_from	int(11)	NO		0	
snmp_errors_from	int(11)	NO		0	
ipmi_error	varchar(128)	NO			
snmp_error	varchar(128)	NO			
jmx_disable_until	int(11)	NO		0	
jmx_available	int(11)	NO		0	
jmx_errors_from	int(11)	NO		0	
jmx_error	varchar(128)	NO			
name	varchar(64)	NO	MUL		

flags	int(11)	NO	0	
templateid	bigint(20) unsigned	YES	MUL	NULL
+-----+-----+-----+-----+-----+				

下面进行具体说明。

- **hostid** : 唯一标识 Host 在 Zabbix 及数据库的 id。不同表之间的关联也是用的 id。和这个类似, Zabbix 中任意一种资源都有自己的 id, 比如 itemid、groupid 等。
- **proxy_hostid** : 如果使用了“Proxy-Server”架构, 这个字段表示的就是监控这台机器的 Proxy 的 hostid。有一点需要注意, 每个 Proxy 在 Hosts 表里有两条 (其他 Host 只有一条记录): 一条是和普通机器一样的, 作为被监控机器的记录; 另一条记录是作为 Proxy 的记录, 其 ip 字段的值为“0.0.0.0”。proxy_hostid 中的值就是 Proxy 记录中的 hostid。如果有一台 Proxy 的 ip 为 1.2.3.4, 那么在 Hosts 表里有两条记录: 一个是 ip 为“1.2.3.4”的记录, hostid 为“1”; 另一个是 ip 为“0.0.0.0”的记录, hostid 为“2”。在这个背景下, 有一台机器的 proxy 是之前提到的那台机器, 那么它在 Hosts 表中的 proxy_hostid 的值为“2”。
- **host** : 机器的 hostname。注意, 在 1.8.8 (即我们使用的) 版本的 Zabbix 中, 如果有两台 hostname 一样的机器, 那么 Zabbix 会 crash 直接退出。
- **dns** : DNS 名称。
- **useip** : 是否用 ip 监控。
- **port** : 监控使用的端口。
- **status** : 机器目前的状态。“0”为正常监控, “1”为 disable, “2”不清楚, 从数据库里找不到 status 为“2”的机器。这好像和 Zabbix 自身的一个 host available 检查有关。“3”表示是个 Template。
- **disable_util、error、available、errors_from** (ipmi_disable_util, ipmi_error... 和 snmp_disable_util... 都是此类): 这几个都是 Zabbix Poller 会去修改的值。当 poller 在第一次取不到值 (根据值的类型不同会更新相应的列, Item 类型为 SNMP 就会更新 snmp_XXX, 默认为“zabbix”类型) 的时候, 会等 15 秒 (CONFIG_UNREACHABLE_DELAY) 来重试, 并且日志会显示“first network error”。如果 15 秒后依然取不到值, zabbix 会在数据库更新这个 Host 取不到值的信息, 并且日志里显示“another network error”。
- **lastaccess**: 这一列是专门为 Proxy 准备的。lastaccess 表示的是 Proxy 最后一次工作的时间。这里的“工作”是指 Zabbix Server 收到 Proxy 数据。

- ◎ inbytes, outbytes : 1.8.8 的代码中没有找到使用这两个字段的代码, 估计是 Zabbix 以后会使用的。
- ◎ useipmi, ipmi_* : 使用 IPMI 时的参数。不展开说。
- ◎ snmp_* : 同上, SNMP 参数。
- ◎ maintenanceid, maintenance_* : 这和 Zabbix 另一个机制 Maintaince 有关, 用于使 Host 置于维护状态而不会报警。

前面只讲了 Hosts 一张表, 所以这里只能介绍一些针对 Host 的操作。

更新机器的 proxy。找到 proxy 的 hostid, 更新对用 host 的 proxy_hostid, 如下。

```
select hostid from hosts where host='ProxyA' and ip='0.0.0.0'; -- get
hostid: 1234
update hosts set proxy_hostid=1234 where host='Host_To_Update_Proxy';
```

更新 Host 的状态 (enable 或 disable), 如下。

```
update hosts set status='0' where host='Host_To_Enable';
update hosts set status='1' where host='Host_To_Disable';
```

16.3 Items表

Items 表也是 Zabbix 的核心表之一, 它记录了 Item 的所有设置。在 Zabbix 中, 最多的操作就是对于 Items 的, 如添加监控项、删除监控项、更新监控项配置等。这一节中, 一起看下 Item 在数据库存储的表——Items 表。

首先看一下表结构:

Field	Type	Null	Key	Default	Extra
itemid	bigint(20) unsigned	NO	PRI	NULL	
type	int(11)	NO		0	
snmp_community	varchar(64)	NO			
snmp_oid	varchar(255)	NO			

hostid	bigint(20) unsigned	NO	MUL	NULL		
name	varchar(255)	NO				
key_	varchar(255)	NO				
delay	int(11)	NO		0		
history	int(11)	NO		90		
trends	int(11)	NO		365		
status	int(11)	NO	MUL	0		
value_type	int(11)	NO		0		
trapper_hosts	varchar(255)	NO				
units	varchar(255)	NO				
multiplier	int(11)	NO		0		
delta	int(11)	NO		0		
snmpv3_securityname	varchar(64)	NO				
snmpv3_securitylevel	int(11)	NO		0		
snmpv3_authpassphrase	varchar(64)	NO				
snmpv3_privpassphrase	varchar(64)	NO				
formula	varchar(255)	NO		1		
error	varchar(128)	NO				
lastlogsize	bigint(20) unsigned	NO		0		
logtimefmt	varchar(64)	NO				
templateid	bigint(20) unsigned	YES	MUL	NULL		
valuemapid	bigint(20) unsigned	YES	MUL	NULL		
delay_flex	varchar(255)	NO				
params	text	NO		NULL		
ipmi_sensor	varchar(128)	NO			data_type	
int(11)	NO		0			
authtype	int(11)	NO		0		
username	varchar(64)	NO				
password	varchar(64)	NO				
publickey	varchar(64)	NO				
privatekey	varchar(64)	NO				
mtime	int(11)	NO		0		

flags	int(11)	NO		0		
filter	varchar(255)	NO				
interfaceid	bigint(20) unsigned	YES	MUL	NULL		
port	varchar(64)	NO				
description	text	NO		NULL		
inventory_link	int(11)	NO		0		
lifetime	varchar(64)	NO		30		
snmpv3_authprotocol	int(11)	NO		0		
snmpv3_privprotocol	int(11)	NO		0		
state	int(11)	NO		0		
snmpv3_contextname	varchar(255)	NO				

下面是具体说明。

- itemid : Item 的 id。
- type : Item 的 type, 和前端界面配置 Item 的 type 对应。数据库中, 这一列的值是 0 到 17 的数字, 分别代表了不同的类型。
- hostid : Item 所在的 Host 的 hostid。如果该 Item 属于 Template, 那么这里显示的是 templateid。
- name : Item 的名字。
- key_ : Item 的 key。
- delay : 实际就是在配置 Item 时候配置的 “Update Interval”。
- history : 前端配置存储 history 的时间。
- trends : 前端配置存储 trend 的时间。
- status : Item 的状态。
- value_type : Item 配置界面中的 “Type of Information”。
- trapper_hosts : 当 Item 为 Trapper 类型的时候, 记录了允许发送数据的 Host。
- units : Item 配置界面中的 “Units”。
- multiplier : 针对这个 Item 是否启用了 “Custom Multiplier”。
- delta : Item 配置界面中的 “Store value” 配置。
- snmpv3* : snmpv3 开头的都是和 SNMP 相关的配置内容。
- formula : 这里即是页面上设置 Items 时的 “Use custom multiplier” 配置的数字。

- error : Item 的错误信息。
- lastlogsize : 针对 log 和 logrt 类型 Items 使用的, 记录上一次读取时日志文件的大小。
- logtimefmt : 针对 log 和 logrt 有效, 配置的日志中时间的格式。
- templateid : 看名字肯定认为记录的是 templateid, 其实不是。当一个 Host 关联一个 Template 的时候, 会把 Template 上的所有 Item 复制到 Host 上。那么 Host 上的 Item 就有一个存在于 Template 上的父 Item, 这里记录的就是 Host 上 Item 对应的在 Template 上的 Item 的 itemid。
- valuemapid : Item 配置界面中选择的 valuemap 的 id。
- delay_flex : Item 配置界面中“Flexible interval”的内容。
- params : 当需要额外参数 Item 时, 记录这些参数的地方。比如执行 SQL 的 Item 会在这里记录需要执行的 SQL。
- ipmi_sensor : 使用 IPMI 类型 Items 需要设定的参数。
- data_type : Item 配置界面中的“Data type”, 选择 Item 返回的数据类型。
- authtype : 在使用 SSH 类型 Item 时有效, 选择 SSH 登录的类型。
- username : 在使用 SSH 类型 Item 时有效, SSH 登录的用户名。
- password : 在使用 SSH 类型 Item 时有效, SSH 登录的密码。
- publickey : 在使用 SSH 类型 Item 时有效, 但是只有当 authtype 设置为使用 publickey 时才有效。
- privatekey : 在使用 SSH 类型 Item 时有效, 但是只有当 authtype 设置为使用 publickey 时才有效。
- mtime : 对 log 和 logrt 有效, 记录了日志文件修改的时间。
- flags : “0”表示一个普通的 Item, “4”表示的是 discover 生成的 Item。
- interfaceid : 当使用 host 类型 Items 时生效, 用来选择 Host 上不同的 interface。
- port : 使用 SNMP 监控时使用的端口。
- description : Item 配置界面上的“Description”。
- inventory_link : 如果这个 Item 的值要作为 Host 的 Inventory, 这里填写的是对应的 inventoryid。
- state : 当前 item 的状态, “0”表示正常, “1”表示“not supported”状态。

16.4 Trigger在数据库中的结构

Trigger 是 Zabbix 的重要部分,平时在工作中,除了 Host 和 Item,接触最多的就属 Trigger 了。而且 Trigger 相对 Host 和 Item 来说,更加复杂。Host 和 Item 在数据库中对应的 hosts 表和 items 表都是非常平面的表,结构简单,Host 和 Item 的属性在表中一目了然。而 Trigger 在数据库中对应的 triggers 表则相对复杂,它和其他表的关联关系很强,需要仔细分析。

下面看看 triggers 的表结构:

```
mysql> desc triggers;
```

Field	Type	Null	Key	Default	Extra
triggerid	bigint(20) unsigned	NO	PRI	NULL	
expression	varchar(2048)	NO			
description	varchar(255)	NO			
url	varchar(255)	NO			
status	int(11)	NO	MUL	0	
value	int(11)	NO	MUL	0	
priority	int(11)	NO		0	
lastchange	int(11)	NO		0	
comments	text	NO		NULL	
error	varchar(128)	NO			
templateid	bigint(20) unsigned	YES	MUL	NULL	
type	int(11)	NO		0	
state	int(11)	NO		0	
flags	int(11)	NO		0	

经过前面几节对于 hosts 表和 items 表的分析,看到这些字段,应该马上就能猜出是什么意思,笔者相信大多数读者朋友到这里已经有这个能力了。

如果还有读者一下没看出来,可以找 Trigger 看一下,如图 16-1 所示。

Parent triggers Template OS Linux

Name Lack of free swap space on {HOST.NAME}

Expression {HostABC:system.swap.size[pfree].last(0)}<50

Expression constructor

Multiple PROBLEM events generation ☐

Description It probably means that the systems requires more physical memory.

URL

Severity

Enabled ☒

图16-1

从 URL 中可以得到 triggerid 为 13588, 我们从该数据库里把这个 Trigger 取出来, 代码如下。

```
mysql> select * from triggers where triggerid=13588\G;
***** 1. row *****
triggerid: 13588
expression: {13191}<50
description: Lack of free swap space on {HOST.NAME}
url:
status: 0
value: 0
priority: 2
lastchange: 1392535037
comments: It probably means that the systems requires more physical
memory.
error:
templateid: 10012
type: 0
state: 0
flags: 0
```

现在能看明白主要字段的作用了吧。Trigger 的核心是 expression, 即我们定义的 Trigger 的

逻辑。我们选取的 Trigger 的逻辑是：{HostABC:system.swap.size[,pfree].last(0)}<50，但数据库里 expression 字段是 {13191}<50，50 这个阈值是吻合的，那么 13191 是什么东西呢？有的朋友可能猜到了——某个 id，类似 Triggerid 的 id。

再看看图中设置 Trigger 逻辑的部分，如图 16-2 所示。

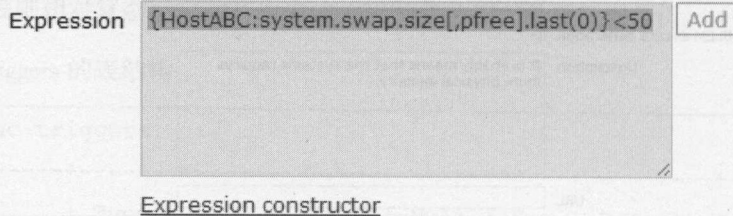


图16-2

我们可以看到这一串逻辑是属于“Expression”属性的，笔者曾经想当然地认为数据库里的 expressions 表就是记录这个的逻辑。但是，当看到 expressions 的内容时，才发现，expressions 表记录的内容，并不是 trigger 里的 expression，而是“Administration”→“General”里的“Regular expressions”，内容如下。

```
mysql> select * from expressions\G;
***** 1. row *****
expressionid: 1
  regexpid: 1
    expression: ^(btrfs|ext2|ext3|ext4|jfs|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|
|hfs|ntfs|fat32|zfs) $
  expression_type: 3
exp_delimiter: ,
case_sensitive: 1
***** 2. row *****
expressionid: 2
  regexpid: 2
    expression: ^lo$
  expression_type: 4
exp_delimiter: ,
case_sensitive: 1
```



```

***** 3. row *****
expressionid: 3
  regexpid: 3
  expression: ^(Physical memory|Virtual memory|Memory buffers|Cached
memory|Swap space)$
  expression_type: 4
  exp_delimiter: ,
case_sensitive: 1
***** 4. row *****
expressionid: 4
  regexpid: 2
  expression: ^Software Loopback Interface
  expression_type: 4
  exp_delimiter: ,
case_sensitive: 1
***** 5. row *****
expressionid: 6
  regexpid: 4
  expression: ext4
  expression_type: 3
  exp_delimiter: ,
case_sensitive: 1
***** 6. row *****
expressionid: 7
  regexpid: 4
  expression: ^ext
  expression_type: 3
  exp_delimiter: ,
case_sensitive: 1

```

再看下配置的 Expressions，如图 16-3 所示。

Name	Expressions
File systems for discovery	1*^(btrfs ext2 ext3 ext4 jfs reiser xfs ufs jfs2 vxfs hfs ntfs fat32 zfs)\${Result is TRUE}
Network interfaces for discovery	1*^lo\$ [Result is FALSE] 2*^Software Loopback Interface[Result is FALSE]
Storage devices for SNMP discovery	1*^(Physical memory Virtual memory Memory buffers Cached memory Swap space)\${Result is FALSE}
testRegularExpression	1*ext4[Result is TRUE] 2*^ext[Result is TRUE]

图16-3

确定 expressions 表是对应这个的了。那我们要寻找的 13191 是什么呢？答案是 function，内容在 functions 表。

```
mysql> select * from functions where functionid=13191;
+-----+-----+-----+-----+-----+
| functionid | itemid | triggerid | function | parameter |
+-----+-----+-----+-----+-----+
|      13191 | 23777 |      13588 | last     | 0          |
+-----+-----+-----+-----+-----+
```

还是看这个例子，Trigger 的 expression 是 {HostABC:system.swap.size[,pfree].last(0)}<50，对比 functions 表，function 和 parameter 列的作用我们都能明白，而 itemid 就是关联的 Item，即这里的 HostABC:system.swap.size[,pfree]。

```
mysql> select itemid,name,key_ from items where itemid=23777;
+-----+-----+-----+
| itemid | name                               | key_ |
+-----+-----+-----+
| 23777 | Free swap space in %              | system.swap.size[,pfree] |
+-----+-----+-----+
```

现在大家应该能明白 Triggers 是怎么设置的了。

我们再看个复杂的：expression 是 “{HostABC:agent.ping.last()}=0 | {HostABC:system.cpu.util[,iowait].last()}=0”，数据库里是下面这样的。

```
mysql> select * from triggers where triggerid=13601\G;
***** 1. row *****
triggerid: 13601
```

```

expression: {13204}=0 | {13205}=0
description: test1
  url:
    status: 0
    value: 0
  priority: 0
lastchange: 1394205728
comments:
error:
templateid: NULL
  type: 0
  state: 0
  flags: 0

```

可以发现，再复杂的 expression 也是由多个 function 拼成的。

16.5 Events表

每当 Zabbix Server 获取到一个数据，它就会检查跟这个 Item 相关的 Trigger，然后无论是否触发 Action，都会生成一个 Event。这一节中，我们看看 Event 在数据库中的存储。首先看 events 表的表结构：

```
mysql> desc events;
```

Field	Type	Null	Key	Default	Extra
eventid	bigint(20) unsigned	NO	PRI	NULL	
source	int(11)	NO	MUL	0	
object	int(11)	NO		0	
objectid	bigint(20) unsigned	NO		0	
clock	int(11)	NO		0	
value	int(11)	NO		0	
acknowledged	int(11)	NO		0	

```
| ns | int(11) | NO | | 0 | | |
+-----+-----+-----+-----+-----+-----+
```

笔者考虑到估计大家对“source”、“object”、“objectid”、“ns”这几个可能有点疑问，其他的应该都能理解。下面进行详细说明。

◎ source : Event 可能由多种源头生成，source 就记录了 Event 是由什么事件而生成的。分为以下几种情况。

- 0 : 由 trigger 生成的 event。
- 1 : 由 discovery rule 生成的 event。
- 2 : 由 agent auto-registration 生成的 event。
- 3 : internal 的 event。

◎ object : 这个字段记录了和 event 关联的 Zabbix 对象。

- 对于 trigger 相关的 events，这里的值只可能是 0。
- 对于 discovery 相关的 event，“1”表示“discovered host”，“2”表示“discovered service”。
- 对于 auto-registration 的 event，这里的值一定是“3”。
- 对于 interval 的 event，“0”表示“trigger”，“4”表示“item”，“5”表示“low-level discovery”。

◎ objectid : 根据前面 object 里的定义，这里可能为 triggerid，也可能是 discovered hostid

◎ ns : 1.8.8 版本的 Zabbix 是没有这个字段的，是在 2.0.0 版本加入这个记录的。因为如果只有 timestamp，那么这个 {ITEM.VALUE} 会发生错乱。所以在 ZBXNEXT-457 中有人提了 bug。地址在 <https://support.zabbix.com/browse/ZBXNEXT-457>。

◎ value : 和 object 字段类似，根据 source 的不同，这里的值有不同的含义。

- 对于 trigger 类型的 event
 - 0 : trigger 的状态为 OK。
 - 1 : trigger 的状态为 PROBLEM。
- 对于 discovery 类型的 event
 - 0 : host 或者 service 正在工作。
 - 1 : host 或者 service 停止工作。

2 : host 或者 service 被侦测到。

3 : host 或者 service 丢失了。

- 对于 internal 类型的 event

0 : normal 状态。

1 : unknown 或者 not supported 状态。

16.6 Triggers和Events生成的规则

在这一节，我们会一起把什么情况下会生成 Trigger 和 Event 研究清楚，这也是让很多人觉得非常棘手的问题，经常有人问，“为什么这个没有产生报警？”

表 16-1 是笔者在看 Zabbix 源代码时发现的，其中表格中字母符号的意思如下。

- “-”表示不可能发生的情况。
- “no”表示没有任何操作。
- “T”表示会更新 Trigger。
- “E”表示会生成一个 Event。
- “(m)”表示这是一个“multiple PROBLEM event”。
- “(e)”表示报错的消息变了。

表16-1

TO \ FROM	OK	OK (?)	PROBLEM	PROBLEM (?)
OK	no	T	T+E	-
OK (?)	T	T (e)	T+E	-
PROBLEM	T+E	-	T (m) + T (e)	T
PROBLEM (?)	T+E	-	T+E (m)	T (e)

第 17 章

History和Trends

History 和 Trends 都是存储历史数据的地方，但是它们有什么区别，一直是大家使用 Zabbix 中的疑惑。这一章中，我们从数据库入手，将这个问题讲清楚。

首先看数据库中与 history 和 trends 相关的表，内容如下。

```
mysql> show tables like '%history%';
```

```
+-----+
```

```
| Tables_in_zabbix (%history%) |
```

```
+-----+
```

```
| history |
```

```
| history_log |
```

```
| history_str |
```

```
| history_str_sync |
```

```
| history_sync |
```

```
| history_text |
```

```
| history_uint |
```

```
| history_uint_sync |
```

```
| proxy_dhistory |
```

```
| proxy_history |
```

```
| user_history |
```

```
+-----+
```

```
mysql> show tables like '%trends%';
```

```

+-----+
| Tables_in_zabbix (%trends%) |
+-----+
| trends                        |
| trends_uint                  |
+-----+

```

其中 trends 表是比较简单的，一共两个表，其中 trends_uint 表示的是 unsigned int，即这两张表的功能是一样的，只是存储的数据类型不同。

17.1 sync字段的含义

history 表比较多，但从表面能够很清楚地看出：大多数 history 表只是分了不同的数据类型。比如 str 表示字符串，log 表示 log 文件类型。其中比较特别的是 sync 属性，sync 一般是“synchronize”的缩写，我们研究下是在什么地方使用的。首先看 grep 源代码，具体如下。

```

$ grep -r 'history_sync' *
libs/zbxdbcache/dbcache.c: * for writing float-type items into history/
history_sync tables.      *
libs/zbxdbcache/dbcache.c: const char *ins_history_sync_sql = "insert
into history_sync (nodeid,itemid,clock,ns,value) values ";
libs/zbxdbcache/dbcache.c:zbx_strcpy_alloc(&sql, &sql_alloc, sql_offset,
ins_history_sync_sql);
libs/zbxdbcache/dbcache.c: zbx_strcpy_alloc(&sql, &sql_alloc, sql_
offset, ins_history_sync_sql);
libs/zbxdbhhigh/dbschema.c: {"history_sync", "id", ZBX_HISTORY_SYNC,
libs/zbxdbhhigh/dbschema.c:CREATE TABLE history_sync (\n\
libs/zbxdbhhigh/dbschema.c:CREATE INDEX history_sync_1 ON history_sync
(nodeid,id);\n\
libs/zbxdbhhigh/dbschema.c:CREATE TABLE 'history_sync' (\n\
libs/zbxdbhhigh/dbschema.c:CREATE INDEX 'history_sync_1' ON 'history_
sync' ('nodeid','id');\n\
libs/zbxdbhhigh/dbschema.c:CREATE TABLE history_sync (\n\
libs/zbxdbhhigh/dbschema.c:CREATE INDEX history_sync_1 ON history_sync

```

```

(nodeid,id);\n\
libs/zbxdbhigh/dbschema.c:CREATE SEQUENCE history_sync_seq\n\
libs/zbxdbhigh/dbschema.c:CREATE TRIGGER history_sync_tr\n\
libs/zbxdbhigh/dbschema.c:BEFORE INSERT ON history_sync\n\
libs/zbxdbhigh/dbschema.c:SELECT history_sync_seq.nextval INTO :new.id
FROM dual;\n\
libs/zbxdbhigh/dbschema.c:CREATE TABLE history_sync (\n\
libs/zbxdbhigh/dbschema.c:CREATE INDEX history_sync_1 ON history_sync
(nodeid,id);\n\
libs/zbxdbhigh/dbschema.c:CREATE TABLE history_sync (\n\
libs/zbxdbhigh/dbschema.c:CREATE INDEX history_sync_1 ON history_sync
(nodeid,id);\n\
libs/zbxdbupgrade/dbupgrade.c: return DBmodify_proxy_table_id_field
("history_sync");
libs/zbxdbupgrade/dbupgrade.c: return DBdrop_index("history_sync",
"id");

```

通过思考, dbschema 中的 CREATE 和 SELECT 应该不是我们需要的地方, 可能会是在 dbcache.c 中, 把包含 “history_sync” 的地方都拿出来看一下, 具体如下。

```

Line 1342: * for writing float-type items into history/history_sync
tables.      *
Line 1349:const char*ins_history_sync_sql = "insert into history_sync
(nodeid,itemid,clock,ns,value) values ";
Line 1349:const char*ins_history_sync_sql = "insert into history_sync
(nodeid,itemid,clock,ns,value) values ";
Line 1378:zbx_strcpy_alloc(&sql, &sql_alloc, sql_offset, ins_history_
sync_sql);
Line 1390:zbx_strcpy_alloc(&sql, &sql_alloc, sql_offset, ins_history_
sync_sql);

```

仔细研究了一下, 发现并不能看出 sync 表的用途, 于是再返回去, 会发现这样一行:

```

libs/zbxdbhigh/dbschema.c: { "history_str_sync", "id", ZBX_HISTORY_SYNC,

```


这一行的“ZBX_HISTORY_SYNC”是全部大写的,这种命名方式似乎有点可以挖掘的地方。我们来 grep 看看,结果如下。

```
$ grep -r 'ZBX_HISTORY_SYNC' *
libs/zbxdbhigh/dbschema.c: {"history_sync", "id", ZBX_HISTORY_SYNC,
libs/zbxdbhigh/dbschema.c: {"itemid",NULL,"items", "itemid", 0, ZBX_
TYPE_ID, ZBX_NOTNULL |
libs/zbxdbhigh/dbschema.c:{"clock","0",NULL,NULL,0,ZBX_TYPE_INT,ZBX_
NOTNULL|ZBX_HISTORY_SYNC
libs/zbxdbhigh/dbschema.c:{"value","0.0000",NULL,NULL,0,ZBX_TYPE_FLOAT,
ZBX_NOTNULL | ZBX_HIS
libs/zbxdbhigh/dbschema.c:{"ns","0",NULL,NULL,0,ZBX_TYPE_INT,ZBX_NOTNULL
| ZBX_HISTORY_SYNC, 0},
libs/zbxdbhigh/dbschema.c:{"history_uint_sync","id",ZBX_HISTORY_SYNC,
libs/zbxdbhigh/dbschema.c:{"itemid",NULL,"items","itemid",0,ZBX_TYPE_ID,
ZBX_NOTNULL |
libs/zbxdbhigh/dbschema.c:{"clock","0",NULL,NULL,0,ZBX_TYPE_INT, ZBX_
NOTNULL | ZBX_HISTORY_SYNC
libs/zbxdbhigh/dbschema.c:{"value","0",NULL,NULL,0,ZBX_TYPE_UINT,ZBX_
NOTNULL | ZBX_HISTORY_SYNC
libs/zbxdbhigh/dbschema.c:{"ns","0",NULL,NULL,0,ZBX_TYPE_INT,ZBX_NOTNULL
| ZBX_HISTORY_SYNC, 0},
libs/zbxdbhigh/dbschema.c:{"history_str_sync","id",ZBX_HISTORY_SYNC,
libs/zbxdbhigh/dbschema.c:{"itemid",NULL, "items","itemid",0,ZBX_TYPE_
ID,ZBX_NOTNULL |
libs/zbxdbhigh/dbschema.c:{"clock","0",NULL,NULL,0,ZBX_TYPE_INT,ZBX_
NOTNULL | ZBX_HISTORY_SYNC
libs/zbxdbhigh/dbschema.c:{"value","",NULL,NULL,255,ZBX_TYPE_CHAR,ZBX_
NOTNULL | ZBX_HISTORY_SYNC
libs/zbxdbhigh/dbschema.c:{"ns","0",NULL,NULL,0,ZBX_TYPE_INT,ZBX_NOTNULL
| ZBX_HISTORY_SYNC, 0},
zabbix_server/nodewatcher/history.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC))
```

```

zabbix_server/nodewatcher/history.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC) && 0 == (table->fields[f].flags & ZBX_HIST
zabbix_server/nodewatcher/history.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC))
zabbix_server/nodewatcher/history.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC))
zabbix_server/nodewatcher/history.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC) && 0 == (table->fields[f].flags &
zabbix_server/nodewatcher/history.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC))
zabbix_server/nodewatcher/history.c:if (0 == (t->flags & (ZBX_HISTORY |
ZBX_HISTORY_SYNC)))
zabbix_server/trapper/nodehistory.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC))
zabbix_server/trapper/nodehistory.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC) && 0 == (table->fields[f].flags & ZBX_HIST
zabbix_server/trapper/nodehistory.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC))
zabbix_server/trapper/nodehistory.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC) && 0 == (table->fields[f].flags & ZBX_HIST
zabbix_server/trapper/nodehistory.c:if (0 != (table->flags & ZBX_HISTORY_
SYNC) && 0 == (table->fields[f].flags & ZBX_HIST
zabbix_server/trapper/nodehistory.c:if (NULL != table && 0 == (table-
>flags & (ZBX_HISTORY | ZBX_HISTORY_SYNC)))
zabbix_server/trapper/nodehistory.c:if (NULL != table && 0 != (table-
>flags & ZBX_HISTORY_SYNC))

```

grep 出来的代码中，大部分都是类似 `0 != (table->flags & ZBX_HISTORY_SYNC)` 的，这是在判断表是不是 `ZBX_HISTORY_SYNC`。这时，我们会发现，包含“`ZBX_HISTORY_SYNC`”的文件除了 `dbschema.c` 外，就是 `nodewatcher/history.c` 和 `nodehistory.c`，所以，这个表是在 Master-Child 架构中使用的。

我们先看 `nodewatcher/history.c`，使用了 `ZBX_HISTORY_SYNC` 的 function 及其说明如下。

- ◎ process_history_table_data : process new history data ; 处理新的历史数据。
- ◎ process_history_tables : process new history data from tables with ZBX_HISTORY* flags, 处理带有 ZBX_HISTORY* 标识的表的数据。

再看 history.c 的主函数 main_historysender, 它的说明是 “periodically sends historical data to master node”, 即 “周期性地将历史数据从子节点发送到父节点。”

这时应该能明白了: 带有 sync 后缀的 history 表, 是用作 Master-Child 同步数据的。那我们在看 history 和 trends 的区别的时候, 把 sync 相关的表略去了。

总结一下, history 和 trends 表在数据库中分别有好几张表来对应不同的类型, 它们都是一样的, 所以简单来说, 我们可以把它们看成两张表: history 表和 trends 表。

17.2 history和trends的区别

上一小节中, 我们主要把 history 在数据库中的几个表刨根问底了一番, 把数据库中的相关表分为了两类——history 表和 trends 表。

它们的相同点是都存储历史数据, 不同点在于存储数据的粒度不同。每次 Zabbix 接收到 Item 的数据以后, 会将其存入 history 表。下面是 history 表结构。

```
mysql> desc history;
```

Field	Type	Null	Key	Default	Extra
itemid	bigint(20) unsigned	NO	MUL	NULL	
clock	int(11)	NO		0	
value	double(16,4)	NO		0.0000	
ns	int(11)	NO		0	

这是个很简单的结构, “clock”和“ns”保存接收 Item 的时间, “itemid”唯一标识了 Item, “value”为接收到的数据。

而 trends 表的作用是将 history 表的数据以小时为维度进行归档。它会针对每一个 itemid, 计算每个小时的最小值、最大值和平均值, 下面是 trends 表结构。

```
mysql> desc trends;
```

Field	Type	Null	Key	Default	Extra
itemid	bigint(20) unsigned	NO	PRI	NULL	
clock	int(11)	NO	PRI	0	
num	int(11)	NO		0	
value_min	double(16,4)	NO		0.0000	
value_avg	double(16,4)	NO		0.0000	
value_max	double(16,4)	NO		0.0000	

其中，num 字段表示了该小时使用了多少数据来计算最小值、最大值和平均值。

17.3 housekeeper和trends表

history 表的作用就是存储所有 Item 的历史数据。前文介绍过，衡量 Zabbix 的性能有一个重要的指标，就是 VPS。我们看一下如果 VPS 为 1000，即每秒 Zabbix 要处理 1000 个数据，每一个数据都会在 history 表中有一行，而一天有 86400 秒，那么每天就是 86400000 行。如果一直这样无限膨胀下去，history 表的性能会非常差，从而拖累整个 Zabbix 数据库的性能。

对于这个问题，Zabbix 提供了两方面的解决办法：一种是 housekeeper 机制，另一种是 trends 机制。

housekeeper 非常简单，就是定期删除 history 表中的数据。对于小规模 Zabbix 来说，这是个省时省力的方法，但对于大规模 Zabbix 数据库，使用 housekeeper 效率非常差，特别是 InnoDB 引擎的 MySQL，因为大表的删除非常慢。针对规模很大的 Zabbix，建议每周 truncate 一次与 history 相关的表，大家可能担心这么暴力的删除会影响数据完整性，看完下面 trends 表的作用，大家就明白了。

下面看看 trends 在 Zabbix 的用处。其实在 Zabbix 中，trends 只在很少几个地方出现，最重要的就是在 Graph 中了，如图 17-1 所示。

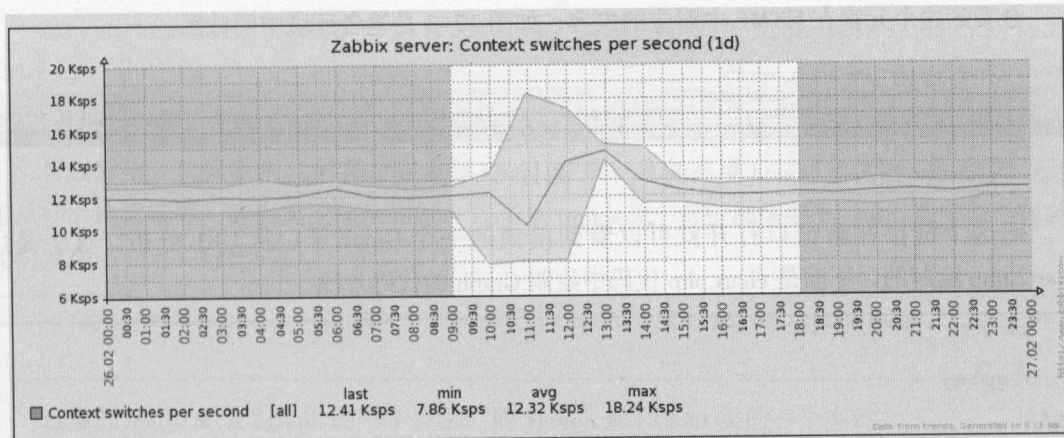


图 17-1

图中对于某个时间点，会有三个数据，从小到大分别是这个时间点的最小值、平均值和最大值。这里调用的就是 trends 表的数据。仔细想一想，history 的数据真的需要保存这么久吗？对于上个星期的数据，真的要精确到每一分钟吗？答案是不需要的，对于时间越久的数据，我们需要的粒度就越粗。

17.4 Graph对于history和trends的选择

在这一节中，我们会分析 Graph 选择从 history 表选数据，还是从 trends 表选数据。

首先先看看图 17-1 是怎么画出来的，我们先进入一个 Item 的 Graph（注意，这里不要选择“Monitoring”→“Graph”中的 graph，而要选择“Monitoring”→“Latest Data”中某个 Item 的 Graph），这里使用了 chrome 的查看元素功能，右键单击页面上的 Graph，选择“查看元素”。如图 17-2 所示。

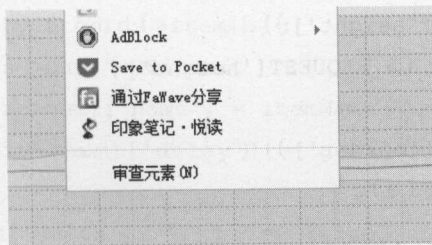


图 17-2

会看到这个元素在 HTML 中对应的内容，如图 17-3 是笔者环境中的截图。

```
<div class="center" id="graph_cont1">
  </div>
```

图17-3

把这个链接在新窗口打开就可以看到单独的一个 Graph 了。从 URL 分析，这个是由 chart2.php 绘制的，下面是 chart.php 代码中绘制 Graph 的代码片段。

```
/*
 * Display
 */
$timeline = CScreenBase::calculateTime (array (
    'profileIdx' => get_request ('profileIdx', 'web.screens'),
    'profileIdx2' => get_request ('profileIdx2'),
    'updateProfile' => get_request ('updateProfile', true),
    'period' => get_request ('period'),
    'stime' => get_request ('stime')
));
$graph = new CChart ();
$graph->setPeriod ($timeline['period']);
$graph->setSTime ($timeline['stime']);
if (isset ($_REQUEST['from'])) {
    $graph->setFrom ($_REQUEST['from']);
}
if (isset ($_REQUEST['width'])) {
    $graph->setWidth ($_REQUEST['width']);
}
if (isset ($_REQUEST['height'])) {
    $graph->setHeight ($_REQUEST['height']);
}
if (isset ($_REQUEST['border'])) {
    $graph->setBorder (0);
}
```

```
$graph->addItem($_REQUEST['itemid'], GRAPH_YAXIS_SIDE_DEFAULT, CALC_FNC_
ALL);
$graph->draw();
```

可以看出,大多数的代码都是在设置 Graph 参数,比如高度 height 和宽度 width 等。那数据从什么地方来呢?数据无论是从 history 还是从 trends 来,肯定都需要 itemid。按照这个思路,我们发现了如下一行代码。

```
$graph->addItem($_REQUEST['itemid'], GRAPH_YAXIS_SIDE_DEFAULT, CALC_FNC_
ALL);
```

这里的 addItem 是 \$graph 的一个方法,而 \$graph 是 CChart 类的一个实例。

```
$graph = new CChart();
```

找到 CChart 定义的地方,如下。

```
$ grep -r CChart *
chart.php:$graph = new CChart();
chart2.php:$graph = new CChart($dbGraph[ 'graphtype' ]);
chart3.php:$graph = new CChart(get_request( 'graphtype' , GRAPH_TYPE_
NORMAL));
include/classes/class.cchart.php:class CChart extends CGraphDraw {
```

接着看 class.cchart.php 中的 addItem 方法。

```
public function addItem($itemid, $axis = GRAPH_YAXIS_SIDE_DEFAULT, $calc_
fnc = CALC_FNC_AVG, $color = null, $drawtype = null, $type = null) {
    if ($this->type == GRAPH_TYPE_STACKED) {
        $drawtype = GRAPH_ITEM_DRAWTYPE_FILLED_REGION;
    }
    $item = get_item_by_itemid($itemid);
    $this->items[$this->num] = $item;
    $this->items[$this->num]['name'] = itemName($item);
    $this->items[$this->num]['delay'] = getItemDelay($item['delay'],
$item['delay_flex']);
    if (strpos($item['units'], ',') !== false) {
        list($this->items[$this->num]['units'], $this->items[$this->
```

```

>num]['unitsLong']) = explode(',', $item['units']);
    }
    else {
        $this->items[$this->num]['unitsLong'] = '';
    }
    $host = get_host_by_hostid($item['hostid']);
    $this->items[$this->num]['hostname'] = $host['name'];
    $this->items[$this->num]['color'] = is_null($color) ? 'Dark Green' :
    $color;
    $this->items[$this->num]['drawtype'] = is_null($drawtype) ? GRAPH_
    ITEM_DRAWTYPE_LINE : $drawtype;
    $this->items[$this->num]['axisside'] = is_null($axis) ? GRAPH_YAXIS_
    SIDE_DEFAULT : $axis;
    $this->items[$this->num]['calc_fnc'] = is_null($calc_fnc) ? CALC_
    FNC_AVG : $calc_fnc;
    $this->items[$this->num]['calc_type'] = is_null($type) ? GRAPH_ITEM_
    SIMPLE : $type;
    if ($this->items[$this->num]['axisside'] == GRAPH_YAXIS_SIDE_LEFT) {
        $this->yaxisleft = 1;
    }
    if ($this->items[$this->num]['axisside'] == GRAPH_YAXIS_SIDE_RIGHT) {
        $this->yaxisright = 1;
    }
    $this->num++;
}

```

可以从头看 addItem 方法的代码，它的工作是将 Items 表中的数据取出来。那还是没有找到是从 history 表还是从 trends 表来的，只能看看 draw() 方法了。draw() 方法比较长，这里就不全部贴出来了，大家可以看下代码，大致了解下 draw() 方法的流程。

在浏览完 draw() 方法后，可以知道 draw() 方法是根据 Item 的数据来绘制每一个点，而 Item 数据的获取，就在 draw 方法的一开始，如下。

```

$this->selectData();

```

我们看看同样在 class.cchart.php 中的 selectData 方法，找到下面这样一行。

```
if (($real_item['history'] * SEC_PER_DAY) > (time() - ($this->from_time +
$this->period / 2)) && ($this->period / $this->sizeX) <= (ZBX_MAX_TREND_DIFF
```

这里有两条逻辑，加上代码中的注释，整理如下。

(1) $(\$real_item['history'] * SEC_PER_DAY) > (time() - (\$this->from_time + \$this->period / 2))$, 注释为“should pick data from history or trends”，即“是从 history 还是 trends 获取数据”。

(2) $(\$this->period / \$this->sizeX) <= (ZBX_MAX_TREND_DIFF / ZBX_GRAPH_MAX_SKIP_CELL)$, 注释为“is reasonable to take data from history?”即“从 history 获取数据是否可行”。

它们的含义分别如下。

(1) Item 保存 history 的天数一共的秒数大于当前时间减去 Graph 开始的时间和一半持续时间的和，即 $items_history_days * 86400 > (now - (graph_start + 0.5 * duration))$ 。

(2) Graph 的持续时间除以 X 轴的长度（像素）小于等于 $ZBX_MAX_TREND_DIFF / ZBX_GRAPH_MAX_SKIP_CELL$ 。其中持续时间除以 X 轴的长度表示 X 轴每一个像素上显示的时间长度。而 $ZBX_GRAPH_MAX_SKIP_CELL$ 表示当多少个像素上没有数据的时候，会显示一个断点。组合起来，这个逻辑就是：每一个像素显示的时间长度小于等于断点时每个像素的时间长度。

总的来说，除了 history 表还使用 trends 表来绘制 Graph，就是为了防止一个像素点的数据过多或者过少而影响数据的展示。

第 18 章

Zabbix和数据库交互详解

Zabbix 的核心是数据与数据库的交互，即增删改查“CRUD”——Create、Read、Update 和 Delete。Zabbix 的设计思想是把所有都落地到数据库，比如已经了解到的报警的规则等，同时，数据库也是 Zabbix 性能优化的重中之重。在本章中，主要向大家剖析 Zabbix 是如何同数据库进行数据交换的，希望通过这一章的学习，大家能对 Zabbix 的后端有所了解。

数据库相关的代码有两部分：一个是 `src/libs/zbxdb`，另一个是 `src/libs/zbxdbhigh`。前者是较为底层的对于数据库的封装，而后者是对前者根据 Zabbix 模型做的封装。

18.1 include/zbxdb.h

`zbxdb.h` 中定义了很多后面要使用的变量和宏。这是 Zabbix 的代码风格，尽量避免程序中出现“Magic number”。“Magic number”指代码中突兀的数字，比如“`status = 1`”，这个“1”就是“Magic number”。对于这种情况，Zabbix 先会在头文件中定义“`#define CONNECT_SUCCESS 1`”，然后在代码里用宏去代替“1”，即“`status = CONNECT_SUCCESS`”。

`zbxdb.h` 中定义了很多宏，这里挑选了常用的来列举，具体如下。

```
◎ #define ZBX_DB_OK      0
◎ #define ZBX_DB_FAIL    -1
◎ #define ZBX_DB_DOWN    -2
◎ #define ZBX_MAX_SQL_SIZE 262144 /* 256KB */
```

这些很容易理解，Zabbix 默认支持 SQL 的长度是 256K。

Zabbix 后端支持多种数据库，那在代码中如何区分不同的数据库呢？可以看下面的例子。

```
#if defined(HAVE_MYSQL)
# include "mysql.h"
# include "errmsg.h"
# include "mysqld_error.h"
# define DB_ROW          MYSQL_ROW
# define DB_RESULT       MYSQL_RES *
# define DBfree_result   mysql_free_result
#elif defined(HAVE_ORACLE)
# include "oci.h"
typedef struct
{
    OCIEnv      *envhp;
    OCIError     *errhp;
    OCISvcCtx    *svchp;
    OCIServer    *srvhp;
    OCISmt       *stmthp; /* the statement handle for execute operations */
}
zbx_oracle_db_handle_t;
# define DB_ROW         char **
# define DB_RESULT       ZBX_ORACLE_DB_RESULT *
# define DBfree_result   OCI_DBfree_result
typedef struct
{
    OCISmt       *stmthp; /* the statement handle for select operations */
    int          ncolumn;
    DB_ROW       values;
    ub4          *values_alloc;
    OCILobLocator **clobs;
}
ZBX_ORACLE_DB_RESULT;
```

```
void    OCI_DBfree_result (DB_RESULT result);
ub4     OCI_DBserver_status ();
```

这段代码是分别针对 MySQL 和 Oracle 的，根据不同的数据库，引入了不同的头文件。Zabbix 的代码中使用 “#if defined” 或者 “#elif defined” 来表示不同的情况。

18.2 zbxdb/db.c

db.c 是针对数据库层做的最底层的封装，在 db.c 中一共定义了很多方法，根据 Zabbix 代码中的注释，我们一起去了解下这些方法的作用。

1. zbx_db_connect：连接数据库。

2. zbx_db_begin：开始一个事务，这个方法其实非常简单，就是在拼接一个大 SQL。比如对于 MySQL 来说，使用这个方法，会在 SQL 最前面加上 “begin”。

3. zbx_db_commit：和 zbx_db_begin 类似，commit 是一个事务，直接在数据库中执行 “commit”。

4. zbx_db_rollback：回滚事故，相当于执行 “rollback”。

5. zbx_db_vselect：执行 select。

6. zbx_db_bytea_escape：将二进制字符串转换为以 “\0” 结尾的字符串。

7. zbx_db_bytea_unescape：与 zbx_db_bytea_escape 相反，将字符串转换为二进制字符串。

8. zbx_db_get_escape_string_len：返回以 “\0” 结尾的字符串的长度。

9. zbx_db_escape_string：将字符串里的转义符号转化为数据库支持的类型。

10. zbx_db_dyn_escape_string：和 zbx_db_escape_string 类似，只是不使用指针返回结果，而是使用方法的返回值。“dyn” 即为 “dynamic” 的意思。zbx_db_escape_string (const char *src, char*dst, size_t len) 需要将结果所在的内存空间分配好，并将其指针传递给 zbx_db_escape_string，而 zbx_db_dyn_escape_string 则只需要传入一个字符串：zbx_db_dyn_escape_string (const char*src) 即可，结果内存空间的分配都由它完成。zbx_db_dyn_escape_string 的代码非常简单，具体如下。

```
size_t len;
char    *dst = NULL;
```



```
len = zbx_db_get_escape_string_len(src);
dst = zbx_malloc(dst, len);
zbx_db_escape_string(src, dst, len);
```

11. `zbx_db_dyn_escape_string_len` : 同上。

12. `zbx_db_get_escape_like_pattern_len` : 获取数据库查询中 LIKE 的字段长度。

13. `zbx_db_escape_like_pattern` : 将 LIKE 中需要转义的符号前加上 “\” 进行转义。

14. `zbx_db_dyn_escap_like_pattern` : 和 `zbx_db_escape_like_pattern` 类似。

上面列举的都是 Zabbix 源码中带有注释的方法，其他还有很多没有注释的，读者可以自行查看。

18.3 zbxdbhigh

前面介绍了 Zabbix 对数据库的基本操作，本节向大家介绍的是 Zabbix 在这些对数据库基础操作上的封装，比如从数据库查询数据的操作。上一节的代码距离 Zabbix 的实际操作稍远，这一节的操作和 Zabbix 的业务更加紧密。

在 `src/libs/zbxdbhigh` 中，一共有下面这些 c 文件（h 文件认为是头文件，具体功能都是在 c 文件中定义）。

- ◉ `db.c` : 大多数方法都在这里，内容很杂，既有对 Trigger 的操作，又有对 Host 的操作。后文会详细说明。
- ◉ `dbschema.c` : 数据库 schema。
- ◉ `discovery.c` : 关于 discovery 封装的方法。
- ◉ `lld.c`、`lld_common.c` 等以 `lld` 开头的 c 文件 : 关于 low-level discovery 封装的方法。
- ◉ `odbc.c` : 关于 ODBC 相关的封装。
- ◉ `proxy.c` : 关于 Zabbix Proxy 的方法的封装。

`db.c` 中定义的方法如下。

(1) `__DBnode` : 和 Zabbix 的分布式架构有关。在 Master-Child 架构中，每一个 Child 都有自己的 id，而这个 id 是本台 Child 服务器数据库中 id 的开头那一位。这个方法就是根据 id 来限定执行 SQL 的 Hosts。比如在 SQL 后增加 “and hostid between 10000000000000 and

19999999999999999999”这样的条件，就可以限定是在 nodeid 为 1 的 Child 服务器上执行。

(2) DBis_node_id : 判断是否为某个 Child 服务器的 id。

(3) DBconnect : 连接数据库，调用了 zbxdb/db.c 中的连接功能，这里增加了日志的输出。

(4) DBinit : 初始化数据库，调用了 zbxdb/db.c 中的 zbx_db_init。

(5) DBtxn_operation : 当数据库出问题时，处理运行的 SQL。这个方法会作为所有 zbxdbhhigh/db.c 中的入口。比如 DBbegin 调用的就是 DBtxn_operaion (zbx_db_begin)，为的就是当数据库出问题时，能够重试 SQL，直到数据库恢复。

(6) DBbegin : 事务开始，调用 zbxdb/db.c 中的 zbx_db_begin。在 zbxdbhhigh/db.c 中几乎所有的方法都会调用 zbxdb/db.c 中对应的方法。

(7) DBcommit : 事务提交。

(8) DBrollback : 事务回滚。

(9) DBend : 事务结束，如果执行成功则调用 DBcommit，如果失败则调用 DBrollback。

(10) __zbx_DBexecute : 执行 SQL。

(11) DBselect_once : 执行 select 语句，只执行一次，即使数据库报错，也只执行一次。

(12) DBselect : 和 DBselect_once 不同的是，DBselect 碰到数据库问题时会在循环中 sleep 重试。

(13) DBselectN : 执行 select 语句，并且返回最前面的 N 条数据。对于 MySQL 即使用“limit”。

(14) process_trigger : 更新 Trigger，生成 Event。

(15) DBdyn_escape_string : 调用 zbxdb/db.c 相应方法。

(16) DBdyn_escape_string_len : 调用 zbxdb/db.c 相应方法。

(17) DBdyn_escape_like_pattern : 调用 zbxdb/db.c 相应方法。

(18) DBget_nextid : Zabbix 每一个对象都有自己的 id，DBget_nextid 就是生成这个唯一 id 的方法。在 Zabbix 1.8 中，“get nextid”是一个非常大的 bug，会造成非常大的性能问题，在 PPTV 时我们是通过修改代码来解决的。

(19) DBadd_condition_alloc : 生成 SQL 的查询条件。

(20) zbx_host_string : 根据 hostid 查询 Host。

- (21) `zbx_host_key_string` : 根据 `hostid` 查询属于这个 Host 的 Items 的 key。
- (22) `zbx_host_key_string_by_item` : 根据 Item 对象返回这个 Item 属于的 Host 和 Item 的 key。
- (23) `zbx_user_string` : 根据 `userid` 查询 user 的 `name`、`surname`、`alias`。
- (24) `DBsql_id_cmp` : 生成比对 `id` 的语句, 形如 “=123”。
- (25) `DBregister_host` : 自动注册 Host, 生成 Event。
- (26) `DBproxy_register_host` : 针对 Proxy 的自动注册 Host。
- (27) `DBexecute_overflowed_sql` : 当 SQL 长度超过限制时, 将 SQL 拆开执行。
- (28) `DBget_unique_hostname_by_sample` : 当输入参数的 `hostname` 已经在数据库存在的时候, 给出可以使用的例子。
- (29) `DBsql_id_ins` : 构建 `id` 的 insert 语句。
- (30) `DBget_inventory_field` : 根据 `inventory_link` 返回 host 的 `inventory` 的名字。
- (31) `DBget_inventory_field_len` : 根据 `inventory_link` 返回 host 的 `inventory` 的长度。
- (32) `DBselect_uint64` : unsigned integer 64 的 select。
- (33) `get_nodeid_by_id` : 根据 `nodeid` 获取源 `id`。

上面就是 `zbxdbhigh/db.c` 中主要的方法。除了 `db.c` 外, 还有几个文件和 `db.c` 的功能是类似的, 都是基于 `zbxdb/db.c` 实现了更加具体的数据库操作。Zabbix 的其他代码, 不会直接调用 `zbxdb/db.c`, 都是调用这里的封装方法。

第 19 章

Zabbix 2.2新功能介绍

在笔者使用 Zabbix 的时候,版本是 1.8.8。迈入 2.0 版本后,从前端到后端都有了很大的改变,在性能方面也有非常大的改进 (nextid 算法改进)。在 Zabbix 的官方博客中,分 11 个方面介绍了 Zabbix 2.2 的新功能,本书也会基于这个顺序进行介绍。

19.1 数据库自动升级

之前的 Zabbix 升级,需要对数据库打 patch。比如在升级 Zabbix 2.0 的时候,就针对不同数据库提供了 patch 文件。在启动 Zabbix server 之前,先要使用它来更改数据库结构。而从 Zabbix 2.2 开始,这一步已经集成在 Zabbix server 启动的进程中。当 Zabbix server 启动时,进程会检查数据库版本,如果必要的话,就会自动升级数据库。而 Zabbix 前端则会暂时停止工作。

按下来就一起看一下 Zabbix server 进程在自动升级数据库过程中的步骤。

19.1.1 检查数据库版本

Zabbix 在升级之前肯定要判断是否需要升级数据库,在数据库中,dbversion 表就是来做这个工作的,内容如下。

```
mysql> select * from dbversion;  
+-----+-----+  
| mandatory | optional |
```



```

+-----+-----+
| 2020000 | 2020000 |
+-----+-----+

```

如果没有这张表，那么就会检查 config 表中有没有“server_check_interval”这一行。如果存在这一个配置，就会认为当前是在使用 Zabbix 2.0。接着就会添加“dbversion”表，并开始升级数据库。

“server_check_interval”是 Zabbix 2.0 独有的一个字段，如果有这个字段，那么肯定就是 Zabbix 2.0。

19.1.2 mandatory和optional字段

Zabbix 在大版本中，对于数据库的 schema 是不允许有大的变动的，只会做一些不影响前后小版本兼容性的变动，比如索引方面的调整等。在以前，对于这些小的变动，由用户自己选择是否进行操作。

现在在升级 Zabbix 的时候，数据库的升级都是由 Zabbix 自己完成的。对于小版本的数据库方面的变动，Zabbix 会在操作后更新“optional”字段，从而告诉 Zabbix 前端目前应该使用什么样的逻辑来组织数据。

我们来分析下 src/libs/zbxdbupgrade/dbupgrade.c 中的代码，这里就不把分析过程写下来了，直接说明结果。

dbversion 一共有两个步骤：一个是创建 dbversion 表，一个是数据库升级。在创建 dbversion 表的时候，会给一个初始的数据库版本，在升级过程中，则会不断根据操作来更新数据库版本。

首先看看创建 dbversion 表的过程，代码如下。

```

static int      DBcreate_dbversion_table(void)
{
    const ZBX_TABLE    table =
    {
        {"dbversion", "", 0,
        {
            {"mandatory", "0", NULL, NULL, 0, ZBX_TYPE_INT, ZBX_NOTNULL, 0},
            {"optional", "0", NULL, NULL, 0, ZBX_TYPE_INT, ZBX_NOTNULL, 0},
        }
    }
}

```

```

        {NULL}
    }
};

int    ret;
DBbegin();
if (SUCCEED == (ret = DBcreate_table(&table)))
{
    if (ZBX_DB_OK > DBexecute("insert into dbversion (mandatory,
optional) values (%d,%d)",
        ZBX_FIRST_DB_VERSION, ZBX_FIRST_DB_VERSION))
    {
        ret = FAIL;
    }
}
DBend(ret);
return ret;
}

```

其中 ZBX_FIRST_DB_VERSION 为 2010000，是最初始的数据库版本号，它距离我们目前数据库的 2020000 还差了很多，因此在 dbupgrade.c 中，又会有如下的操作。

```

static int    DBpatch_2010197(void)
{
#ifdef HAVE_ORACLE
    return ZBX_DB_OK > DBexecute("update alerts set message_tmp=message")
? FAIL : SUCCEED;
#else
    return SUCCEED;
#endif
}

```

这个方法叫做 DBpatch_2010197，即代表了这个 2010197 的操作，像这样的操作还有很多。下面是调用这些 patch 的地方。

```

DBPATCH_START()
/* version, duplicates flag, mandatory flag */

```

```
DBPATCH_ADD (2010001, 0, 1)
```

```
DBPATCH_ADD (2010002, 0, 1)
```

```
DBPATCH_ADD (2010003, 0, 1)
```

```
...
```

```
DBPATCH_ADD (2020000, 0, 1)
```

最后一步就是版本号为 2020000 的操作，所以我们在数据库中看到的的就是 2020000 这个版本号了。

19.1.3 数据库升级过程

当 Zabbix server 在进行数据库升级的时候，在日志中会有下面类似的内容。

```
current database version (mandatory/optional): 02010008/02010008
```

```
required mandatory version: 02010021
```

```
starting automatic database upgrade
```

```
completed 7% of database upgrade
```

```
completed 15% of database upgrade
```

```
completed 23% of database upgrade
```

```
completed 30% of database upgrade
```

```
completed 38% of database upgrade
```

```
completed 46% of database upgrade
```

```
completed 53% of database upgrade
```

```
completed 61% of database upgrade
```

```
completed 69% of database upgrade
```

```
completed 76% of database upgrade
```

```
completed 84% of database upgrade
```

```
completed 92% of database upgrade
```

```
completed 100% of database upgrade
```

```
database upgrade fully completed
```

从日志里能看到每一个升级数据库的步骤。如果其中任何一步报错，那么 Zabbix server 会退出并且执行一些修复操作。当 Zabbix server 重新启动的时候，会根据 dbversion 表中的数据，继续进行升级操作。

19.1.4 前端提示

如果前端和数据库中的 mandatory 字段数据不符合(optional 忽略),或者不存在 dbversion 表,那么前端界面会显示如图 19-1 所示的数据。

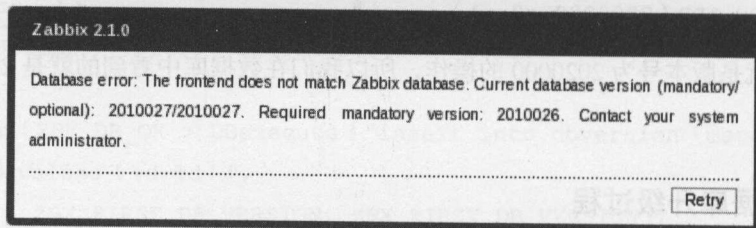


图19-1

比如 Zabbix 2.2 的前端中,在 include/classes/db/DbBackend.php 中的代码如下。

```
public function checkDbVersion() {
    if (!$this->checkDbVersionTable() {
        return false;
    }

    $version = DBfetch(DBselect('SELECT dv.mandatory,dv.optional FROM
dbversion dv'));
    if ($version['mandatory'] != ZABBIX_DB_VERSION) {
        $this->setError(_s('The frontend does not match Zabbix
database. Current database version (mandatory/optional): %d/%d. Required
mandatory version: %d. Contact your system administrator.',
$version['mandatory'], $version['optional'], ZABBIX_DB_VERSION));
        return false;
    }
    return true;
}
```

代码中可以看到判断数据库版本的逻辑 “\$version['mandatory'] != ZABBIX_DB_VERSION”, “ZABBIX_DB_VERSION” 是在 include/defines.inc.php 中定义的,如下。

```
:define ('ZABBIX_DB_VERSION', 2020000);
```


19.2 Web 监控

19.2.1 Web 监控 Template 化

在 Zabbix 2.2 版本之前, Web 监控都绑定在一个特殊的 Host 上, 即不是我们能看见的某个 Host, 这对于维护是非常困难的。比如监控的 URL 和某个 hostname 有关, 如果有 10 台机器服务于一个应用, 服务器名字分别为 HOSTNAME_1 到 HOSTNAME_10, 它们对外服务的 URL 为 http://HOSTNAME_1 到 http://HOSTNAME_10, 那么以前就需要在 Web 监控中添加 10 个监控。如果对这些机器有添加或者减少, 就要去维护 Web 监控的 URL。

在 Zabbix 2.2 中, Web 监控就像 Item 一样, 是绑定在 Host 上的, 而且在配置监控的时候, 可以使用宏来定义 URL。比如上一段中提到的例子, 把 URL 写成 http://{HOST.HOST} 就行了。

而且, 在 Zabbix 2.2 之前, 一个 Web scenario 一定要和一个 Application 关联, 现在也不需要了。

19.2.2 Web 监控重试机制

我们在访问网络, 或者进行 curl 检查的时候, 经常会因为各种各样的网络问题造成访问不成功, 特别是在中国。在 Zabbix 2.2 中, Web 监控增加了重试机制, 大家可能会很奇怪: 这个重试难道不是必须的吗? 其实对于为什么 Zabbix 在以前版本是没有重试机制, 笔者也存在疑惑。设置如图 19-2 所示。

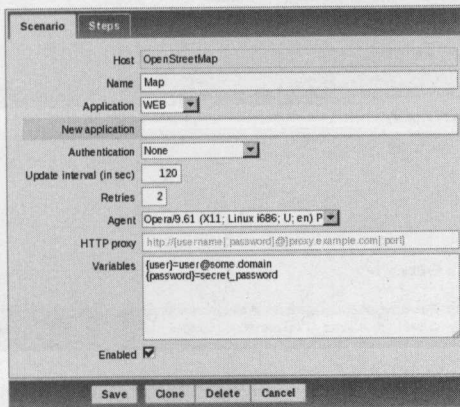


图 19-2

图中的“Retries”设置, 最多能重试 10 次。

19.2.3 使用HTTP代理

HTTP代理大家应该比较熟悉了,我们常用的“Proxy SwitchSharp”就是这种。在监控中,需要经过某个代理去监控一个URL。Squid或者类似的经常用作CDN的监控就需要这样的功能。比如,我们要看各个地区对一个URL的访问情况,就可以使用HTTP代理到各个地区的IDC机房,然后去请求URL,就可以得到这个数据了。

在Zabbix中,Web URL监控使用libcurl,即在Zabbix Server或者Zabbix Proxy的Linux服务器上设置http_proxy环境变量:

```
http_proxy=http://host_name:3278
```

这样做最大的缺点就是这个设置是全局性的,是针对所有Zabbix Server或者Zabbix Proxy执行的Web监控全部生效,而不能只对某一个Web监控生效。所以针对我们的例子,用来测试各个地区对于CDN的访问情况,是很难完成的。

为了解决这个问题,Zabbix 2.2针对每一个URL监控都可以设置使用的代理,如图19-3所示。

The screenshot shows the Zabbix configuration interface for a new monitoring step. The 'Steps' tab is active. The configuration fields are as follows:

- Host: OpenStreetMap
- Name: wiki
- Application: WEB (selected from a dropdown)
- New application: (empty text field)
- Authentication: None (selected from a dropdown)
- Update interval (in sec): 120
- Retries: 1
- Agent: Lynx 2.8.7rel1 (selected from a dropdown)
- HTTP proxy: http://[username[:password]@]proxy.example.com[:port]
- Variables: (empty text area)
- Enabled: ☒

At the bottom, there are buttons for Save, Clone, Delete, and Cancel.

图19-3

19.2.4 URL 监控中使用页面内容作为变量

先从一个例子开始。

有一个邮箱服务，它有以下几个 URL。

(1) `http://localhost/login`

(2) `http://localhost/query?sid=XXX`

(3) `http://localhost/logout?sid=XXX`

在第一步的 login 后，会获取一个 JSON——“{“sid” :” XXX” }”，只有使用这个 sid 才能继续访问 query 和 logout 接口。我们的需求就是要监控这三个 API 的访问情况。

在 Zabbix 2.2 之前，在 URL 监控的 Variables 中只能写一些固定的变量，如果碰到这样的需求就束手无策了，因为 sid 是根据 login 的返回值来设定的。

这样的需求有很多，因为，Web 监控的精髓就是模仿用户的一次完整访问链，从而可以知道每一个步骤是否是正常的。除了上面说的邮箱服务，可能还有聊天工具的接口、购物的多个接口调用等。它们的特点就是需要使用页面中的一些内容作为下一步监控的变量，比如聊天工具接口的监控也是需要类似 sid 的东西，购物的接口需要有具体的某个物品的变量。下面我们看看如何进行类似的操作。

操作很简单，在 Variables 里使用正则表达式来抓取 `sid : {sid}=regex:” sid” :” ([0-9a-z])”` 就可以了。

“regex” 告诉 Zabbix，后面跟着的这串是一个正则表达式，然后将匹配得到的内容赋值给变量 sid。使用虽简单，但也有需要注意的地方，具体如下。

- (1) 正则表达式至少要有有一个子组。
- (2) 如果有多个子组，那么会使用第一个匹配到的子组。
- (3) 支持跨行匹配。
- (4) 大小写敏感。

如果没有子组或者没有匹配到任何东西，那么 Web scenario 这一步将会失败。

19.3 数据映射

数据映射的作用就是把不可读的数据转换成可读的数据。比如“0”表示正常，“1”表示不正常。Zabbix 在 2.2 之前，只支持将数字映射成字符串，如图 19-4 所示。

从 Zabbix 2.2 开始，还增加了对字符串的映射，如将“GOOD”映射为“OK”，“BAD”映射为“ERROR”。在 Zabbix 2.2 之前，如果使用字符串，是会报错的，如图 19-5 所示。

Timestamp	Value
2013.Feb.20 11:59:03	highTemperature (5)
2013.Feb.20 11:58:03	ok (3)
2013.Feb.20 11:57:03	ok (3)
2013.Feb.20 11:56:03	ok (3)

图 19-4

图 19-5

19.4 history和trends存储的代码分析

前面已经介绍过 history 和 trends 在 Zabbix 中的地位，它们是 Zabbix 存储数据的地方。在本节中，我会给大家分析 Zabbix 内部是如何处理这两份数据的。前文中已经提到了，history_str/history_log 之类的表只是 history 对于不同类型值的表，在本节中，我们以基本的 history 和 trends 表来介绍。

Zabbix 写入数据的逻辑入口是 src/libs/zbxdbcache/dbcache.c 中的 DCsync_all() 方法，dbcache 是 Zabbix 将数据保存在内存并且 flush 到数据库的一个功能。

对于代码的分析，我会直接写在代码注释中，以方便大家阅读。

```
static void DCsync_all()
{
    // DCsync_history 的作用是将 history 数据从缓存写入数据库
    DCsync_history(ZBX_SYNC_FULL);
    // DCsync_trends() 的作用是同步 trends 数据
    DCsync_trends();
}
```

可以看到，代码逻辑主要是在 DCsync_history 和 DCsync_trends 中，下面我们分别看下这两段代码。

19.4.1 DCsync_history

下面是截取的 DCsync_history 中比较关键的逻辑代码：

```
intDCsync_history(intsync_type)
{
    /*
        1.      初始化变量
        2.      当 sync_type 为 SYNC_TYPE_FULL 时，解除所有 trigger 对于 items
        的锁定，因为 Zabbix 在处理 trigger 的时候回锁定相关 item
        3.      锁定需要处理的 item 对应的 trigger
        4.      锁定 CACHE
        5.      得到需要处理的 history 的 itemid 列表
        6.      根据 itemid 列表，锁定列表中 item 对应的 trigger
        7.      构建需要处理的 history 列表
    */
    // 更新 history
    DCmass_add_history(history, history_num);
    // 更新 trends
    DCmass_update_trends(history, history_num);
    /*
        1.      解锁之前锁定的 itemid 列表对应的 trigger
        2.      锁定 CACHE
        3.      将处理完的 history 从 CACHE 中移除
        4.      解锁 CACHE
        5.      清理变量
    */
}
```

主要逻辑是 DCmass_add_history 和 DCmass_update_trends。

DCmass_add_history

DCmass_add_history 首先会计算需要更新的 history 中各个类型的 value 有多少，逻辑非常简单，根据每一条 history 的 value_type，根据不同 type（比如 ITEM_VALUE_TYPE_FLOAT 等），增加不同类型的计数器。

得到数量之后，根据类型，调用不同的 `dc_add_history` 的方法（比如 `dc_add_history_dbl`、`dc_add_history_uint` 等）来执行 SQL。代码如下：

```
/* history */
dc_add_history_dbl(history, history_num);
/* history_uint */
dc_add_history_uint(history, history_num);
```

我们跟着代码，再进入 `dc_add_history_dbl`，它的功能非常简单，就是执行 SQL。

```
static void dc_add_history_dbl(ZBX_DC_HISTORY *history, inthistory_num)
{
    zbx_db_insert_add_values(&db_insert, history[i].itemid, history[i].
ts.sec, history[i].ts.ns, history[i].value.dbl);
}
```

其中 `zbx_db_insert_add_values` 就是拼接 insert 的 SQL 了。

DCmass_update_trends

`DCmass_update_trends` 的功能是将 history 写入 trends。

```
static void DCmass_update_trends(ZBX_DC_HISTORY *history, inthistory_num)
{
    /*
        1.      初始化变量
        2.      锁定 trends
    */
    // 根据 history 构建 trend
    DCadd_trend(&history[i], &trends, &trends_alloc, &trends_num);
    // 将 trend 转换为 array
    DCflush_trend(trend, &trends, &trends_alloc, &trends_num);
    /*
        1.      解锁 trends
    */
    // 将 trends 列表插入数据库
    DCflush_trends(trends, &trends_num, 1);
}
```

19.4.2 DCsync_trends

```
static void DCsync_trends()
{
    ...
    while(NULL != (trend = (ZBX_DC_TREND *) zbx_hashset_iter_next(&iter)))
        DCflush_trend(trend, &trends, &trends_alloc, &trends_num);
    UNLOCK_TRENDS;
    DBbegin();
    while(trends_num > 0)
        DCflush_trends(trends, &trends_num, 0);
    DBcommit();
    ...
}
```

两个主要的方法是 DCflush_trend 和 DCflush_trends，前面已经介绍过了，这里就不再介绍。

19.4.3 整个流程

下面我们看看整个流程，如图 19-6 所示。

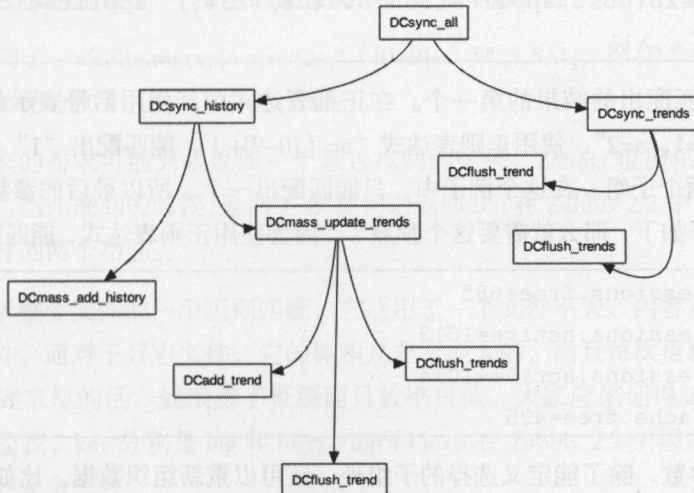


图 19-6

19.5 网页字符串匹配

在开发中都会使用一个 ok.jsp（或者类似的文件），里面会记录一些应用的健康信息。比如记录了“status=ok”，只需要看这个 URL 有没有返回这个字符串就知道应用是否正常了。

Zabbix 很早就提供了一个全新的 Item，叫做 web.page.regexp[]，用来判断返回的 Web 页面是否有包含的字符串，Item 的 key 如下：

```
web.page.regexp[host, <path>, <port>, <regexp>, <length>]
```

但是如果匹配出多个满足条件的字符串，就没办法了，我们不能指定需要返回的是哪一个。而在 Zabbix 2.2 中，增加了一个参数——output。熟悉正则表达式的朋友知道，正则匹配中有一个 subgroup——子组的概念，它用来选择匹配得到的结果中的哪一个。

假设 ok.jsp 返回如下：

```
application.sessions.free=685
application.sessions.active=1013
application.cache.free=425
```

需要写的 key 如下：

```
web.page.regexp[host,application/status,12345,"application.sessions.
active=([0-9]+)","",\1]
```

“\1”表示只要匹配出的结果的第一个。在正则表达式中，使用括号表示的是组的概念。比如对于字符串“a=1, a=2”，使用正则表达式“a=([0-9]+)”，能匹配出“1”和“2”这两个结果，它们分别是两个子组。在这个例子中，只能匹配出一个，所以最后的参数“\1”是可有可无的。但如果例子如下，那么就需要这个参数了，因为使用正则表达式，能匹配出两个结果。

```
application.sessions.free=685
application.sessions.active=1013
application.sessions.active=1014
application.cache.free=425
```

最后的 output 参数，除了能定义选择的子组外，还可以重新组织数据，比如将 output 设置为“\2:\1”，那么返回值就是“1014:1013”，相当于使用“\1”来表示第一个子组。

如果子组不存在，那么就返回空字符串。“\0”表示返回整个字符串。

19.6 日志文件监控

上一小节中,介绍了 Zabbix 对待网页内容的正则匹配,除了在网页内容的扩展,在日志文件的关键字匹配中,也进行了增强。

“`vfs.file.regexp[]`”这个 key 是对于一个文件进行关键字匹配,如果匹配成功,那么就返回一行日志。使用这个 key 的格式为:

```
vfs.file.regexp[file, regexp, <encoding>]
```

在 Zabbix 2.2 中,增加如下两个功能。

- (1) 支持只返回匹配到的内容。
- (2) 支持在文件中的某两行之间的范围内查找匹配。

key 的格式也变为:

```
vfs.file.regexp[file, regexp, <encoding>, <start line>, <end line>, <output>]
```

其中:

- `<start line>` 和 `<end line>` 表示从日志的哪一行开始,到哪一行结束,中间进行匹配。
- `<output>` 的设置和上一小节中的配置类似,这里不做赘述。

我们看个例子: `vfs.file.regexp[/some/file,"([0-9]+) $",,3,5,1]`,即在 `“/some/file”` 的第 3 行到第 5 行寻找出现的数字,返回第一个子组。

有时,大家的需求可能只是返回一个是否找到的结果,Zabbix 也提供了这样的 key: `“vfs.file.regmatchp[]”`,当匹配到的时候,返回 1,没找到则返回 0。在 Zabbix 2.2 中,它也增加了和 `“vfs.file.regexp”` 一样的两个功能。

本节是对于整个文件的一个正则匹配,它适用于一个文件不大、内容不太改变的文件,比如某个配置文件。而对于日志文件,它的体积是非常庞大的,而且每次更新都是在末尾,所以如果从头匹配到末尾的话,就浪费了资源而且效率极低。大家应该记得 Zabbix 有专门针对日志文件的两个监控, key 分别是 `log` 和 `logrt`。`log` 和 `logrt` 在 Zabbix 2.2 中添加了 `“<output>”`,功能和上面提到的完全一致。

19.7 Latest Data局部刷新

这是个非常简单的新特性，但又是非常重要的新特性。在以前使用 Zabbix 的过程中，查看 Latest Data 是最常用的功能了。我们在看的时候，肯定是要展开某个 Application 的，如图 19-7 所示。

[-] Zabbix.org	MySQL (5 Items)		
	MySQL DELETE statements per second	09 May 2013 19:37:03	0 qps
	MySQL INSERT statements per second	09 May 2013 19:37:04	366.13 mpps
	MySQL SELECT statements per second	09 May 2013 19:37:01	6.81 qps
	MySQL UPDATE statements per second	09 May 2013 19:37:02	3.36 qps
	Number of running processes mysqld	09 May 2013 19:35:57	1

图 19-7

每当单击一个 Application 将其打开时，Zabbix 会刷新整个页面，这是个非常奇怪的机制。因为只安装了 Zabbix 2.2，所以笔者怀疑在以前的版本中，单击加号展开一个 Application 是将这个 Application 的名字当做一个参数传给了 PHP，然后就相当于打开了一个新的页面。这个操作真的是噩梦，每次一点就要刷新页面，而且刷新后还要找到之前单击的 Application。从性能上来说，每次刷新页面都要载入重复的数据，对数据库也是重复的开销。

幸好，在 Zabbix 2.2 中，已经把这个给改掉了，单击 Application 展开的时候不会再刷新页面了。从原理上，Zabbix 在打开 Latest Data 页面的时候，已经将数据全部载入了，展开或关闭 Application 都不会载入新的数据。

19.8 动态载入模块

这个是 Zabbix 2.2 的重量级新特性，但是使用的范围还非常窄。

动态载入模块的英文名字是“loadable modules”，它的出现使得我们可以自己编写 Zabbix 的监控项。一般情况下，通过定义“user parameters”是可以做到用户自己写监控脚本，来监控数据的。但是在规模较大或者监控项较多的情况下，就会有问题了。因为当 Zabbix Agent 执行用户自定义的脚本的时候，它是 fork 出一个进程来处理的，这显然对性能是有影响的。而且，它不具备可移植性。比如我写的脚本会依赖我自己的环境，从而直接发送给别人，别人无法使用。

基于上面这些理由, Zabbix 在 2.2 版本迎来了动态载入模块的新功能。一个模块以一个 SO 文件提供。SO 是 shared object 的全称, 可以简单理解为 Python 中被 import 的模块。只要编写符合规范的 SO 文件, 就可以像 Zabbix 内置的那些 Items 一样使用了。

在 zabbix_agentd.conf 和 zabbix_server.conf 中需要配置的参数有 LoadModulePath 和 LoadModule, 它们的用法如下。

```
LoadModulePath=/usr/local/lib/zabbix/agent/
LoadModule=mariadb.so
LoadModule=apache.so
LoadModule=kernel.so
LoadModule=dummy.so
```

mariadb.so 就是我们写的模块了。

毫无疑问动态可载入模块这是一个非常棒的特性, 但是开发这个需要使用 C 语言, 这个对于目前互联网行业的兄弟们来说, 应该算是个比较冷门的语言。而且对于一般使用 Zabbix 的人来说, 使用动态可载入模块在性能上的收获可能远小于找人开发这些模块的成本。我觉得更合适的方法是大家遵从一套开发 Zabbix 监控脚本的规范即可, 除了语言自带的库, 不要去依赖公司相关的库, 这样就能做到监控脚本的通用性了。

一个可载入模块的例子

下面使用在官方博客上提供的一个例子来向大家说明。这里要开发的模块叫做 dummy.so, 目录结构如下。

```
alex@alex:~/path/to/zabbix/src/modules/dummy$ ls -l
-rw-rw-r-- 1 alex alex 9019 Apr 24 17:54 dummy.c
-rw-rw-r-- 1 alex alex 67 Apr 24 17:54 Makefile
-rw-rw-r-- 1 alex alex 245 Apr 24 17:54 README
```

dummy 提供了三个 key, 分别是:

- (1) dummy.ping: 永远返回 “1”。
- (2) dummy.echo[param1]: 返回第一个参数, 比如 dummy.echo[ABC] 返回的是 “ABC”。
- (3) dummy.random[param1, param2]: 返回 param1 到 param2 之间的随机数。

先看 dummy.ping 的实现, 如下。

```
int zbx_module_dummy_ping (AGENT_REQUEST *request, AGENT_RESULT *result)
{
    SET_UI64_RESULT (result, 1);
    return SYSINFO_RET_OK;
}
```

dummy.ping 的逻辑异常简单，大家可能一看这“return SYSINFO_RET_OK”就认为这个值就是返回的“1”了，其实，这个 SYSINFO_RET_OK 是在 include/module.h 中定义的，值为 0。真正设置返回值的是“SET_UI64_RESULT (result,1)”。return 这一句只是告诉 Zabbix 这次调用是正常的。

再看 dummy.echo 的逻辑，如下。

```
int zbx_module_dummy_echo (AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char *param;
    if (1 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT (result, strdup ("Invalid number of parameters"));
        return SYSINFO_RET_FAIL;
    }
    param = get_rparam (request, 0);
    SET_STR_RESULT (result, strdup (param));
    return SYSINFO_RET_OK;
}
```

由于 dummy.echo 需要一个参数，所以开始就要先判断是不是有参数，这里就要使用 AGENT_REQUEST 结构体自带的 nparam 了，这个变量是参数的个数，AGENT_REQUEST 的定义如下：

```
typedef struct
{
    char          *key;
    int           nparam;
    char          **params;
    zbx_uint64_t  lastlogsize;
}
```

```

    int             mtime;
}
AGENT_REQUEST;

```

核心的逻辑是“get_rparam (request,0)”，它是在 module.h 中定义的宏，用来获取参数中的第 0 个参数，定义如下。

```

#define get_rparam (request, num)    (request->nparam > num ? request->
>params[num] : NULL)

```

SET_STR_RESULT 和 return 与前面类似。

最后看看最复杂的 dummy.random，代码如下。

```

int zbx_module_dummy_random (AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char *param1, *param2;
    int from, to;
    if (request->nparam != 2)
    {
        /* set optional error message */
        SET_MSG_RESULT (result, strdup ("Invalid number of parameters"));
        return SYSINFO_RET_FAIL;
    }
    param1 = get_rparam (request, 0);
    param2 = get_rparam (request, 1);
    /* there is no strict validation of parameters for simplicity sake */
    from = atoi (param1);
    to = atoi (param2);
    if (from > to)
    {
        SET_MSG_RESULT (result, strdup ("Incorrect range given"));
        return SYSINFO_RET_FAIL;
    }
    SET_UI64_RESULT (result, from + rand () % (to - from + 1));
    return SYSINFO_RET_OK;
}

```

核心逻辑是以下几个。

```
param1 = get_rparam(request, 0);
param2 = get_rparam(request, 1);
/* there is no strict validation of parameters for simplicity sake */
from = atoi(param1);
to = atoi(param2);
SET_UI64_RESULT(result, from + rand() % (to - from + 1));
```

rand() 是返回一个随机数, 然后再根据 to 和 from 的值取模。再加上 from, 就能获取到 from 到 to 之间的一个随机数了。

19.9 SNMP监控改进

Zabbix 支持很多 SNMP 类型的监控, Zabbix 2.2 对 SNMP 做了一些增强。

19.9.1 SNMPv3相关的增强

对于 SNMPv3, Zabbix 2.2 增加了两个功能: 一个是对 Context 的支持, 另一个是在验证过程中增加了 SHA 和 AES 的验证方式。

SNMPv3 的 context 是用来区分一个 SNMP 后面的多个设备的。比如一个 UPS 工作站连接了很多个 UPS 设备, 但它暴露的只有一个 SNMP 接口, 对于一个监控信息只有一个 OID, 那么这时候可以使用 context 来区分后面的这些 UPS 设备。从 Zabbix 2.2 开始, 在下面这些地方都可以使用 SNMPv3 的 context。

- (1) 普通 Items
- (2) lld 规则
- (3) lld 的 Item 原型
- (4) network 侦测

另外, 原来的 Zabbix 只支持 MD5、DES, Zabbix 2.2 中添加了对 SHA 和 AES 的支持, 如图 19-8 所示。

Check type: SNMPv3 agent

Port range: 161

SNMP OID:

Context name:

Security name:

Security level: authPriv

Authentication protocol: MD5 SHA

Authentication passphrase:

Authentication protocol: DES AES

Privacy passphrase:

Add Cancel

图19-8

19.8.2 SNMP重试和超时机制改进

在 Zabbix 2.2 之前, SNMP 类型的监控是不能自己设置重试次数和超时时间的, Zabbix 会使用 Net-SNMP 库中的默认值——超时时间为 1 秒, 重试 5 次。这样几乎需要 6 秒的时间, 对于 SNMP 网络设备的自动侦测的效率非常低。

从 Zabbix 2.2 开始, 使用“Timeout”参数来设定超时时间, 默认为 3 秒, 而且一旦失败, 则不会重试。

19.9.3 lld的复杂OIDs

在 Zabbix 2.2 之前, 对于一个 OID, Zabbix 只会取其最后一个值来做 lld, 这样对于很长的 index 会发生问题, 比如下面这种。

```
CISCO-POP-MGMT-MIB::cpmDS1ActiveDS0s.6.0
```

```
CISCO-POP-MGMT-MIB::cpmDS1ActiveDS0s.6.1
```

```
CISCO-POP-MGMT-MIB::cpmDS1ActiveDS0s.7.0
```

当侦测“CISCO-POP-MGMT-MIB::cpmDS1ActiveDS0s”这个 OID 的时候, Zabbix 只会创建 0 和 1 两个 OID, 因为上面三条 OID 的最后一个值是“0”、“1”、“0”。而真实情况是“6.0”、“6.1”、“7.0”。在 Zabbix 2.2 中, 会对完整的 OID 进行处理。

第 20 章

Zabbix内置监控项实现

这一章中，我们会解析两个 Zabbix 自带的监控项：system.hostname 和 cpu load。

20.1 system.hostname

从 system.hostname 入手，先 grep 关键字，代码如下。

```
$ grep -r 'system.hostname' *
libs/zbxsystinfo/aix/hostname.c: {"system.hostname",0,SYSTEM_HOSTNAME,NULL};
libs/zbxsystinfo/freebsd/hostname.c: {"system.hostname",0,SYSTEM_
HOSTNAME,NULL};
libs/zbxsystinfo/hpux/hostname.c: {"system.hostname",0,SYSTEM_HOSTNAME,NULL};
libs/zbxsystinfo/linux/hostname.c: {"system.hostname",0,SYSTEM_HOSTNAME,NULL};
zabbix_agent/zabbix_agentd.c:CONFIG_HOSTNAME_ITEM = zbx_strdup (CONFIG_
HOSTNAME_ITEM, "system.hostname");
zabbix_agent/zabbix_agentd.c:zabbix_log (LOG_LEVEL_WARNING, "failed to
get system hostname from [%s] ", CONFIG_HOSTNAME_ITEM);
zabbix_proxy/proxy.c:CONFIG_HOSTNAME_ITEM = zbx_strdup (CONFIG_HOSTNAME_
ITEM, "system.hostname");
```

从 grep 出的结果，我们基本就能确定代码的位置了。从目录来看，Zabbix 对于不同的系统有不同的监控，具体是在 src/libs/zbxsystinfo/SYSTEM_TYPE/hostname.c 中，其中 SYSTEM_TYPE

就是不同的操作系统。我们选择 Linux 的 `hostname.c` 去看一下。

```
#include "sysinfo.h"
#ifdef HAVE_SYS_UTSNAME_H
# include <sys/utsname.h>
#endif

ZBX_METRIC parameter_hostname =
    /* KEY FLAG FUNCTION TEST PARAMETERS */
    {"system.hostname", 0, SYSTEM_HOSTNAME, NULL};

int SYSTEM_HOSTNAME (AGENT_REQUEST *request, AGENT_RESULT *result)
{
    struct utsname name;
    if (-1 == uname (&name))
        return SYSINFO_RET_FAIL;
    SET_STR_RESULT (result, zbx_strdup (NULL, name.nodename));
    return SYSINFO_RET_OK;
}
```

这一段就是 `system.c` 的内容，代码非常简单，`SYSINFO_RET_FAIL` 和 `SYSINFO_RET_OK` 表示一个函数执行成功或者不成功。`SET_STR_RESULT` 是将一个值赋给一个字符串。代码的核心在于引用了 GNU C 库的 `utsname.h`，定义了 `utsname` 结构体，然后从其中获得了 `hostname`——`name.nodename`。

那在前端配置的 `system.hostname` 的时候，Zabbix 怎么知道执行这个代码呢？看下面的代码。

```
ZBX_METRIC      parameter_hostname =
/* KEY          FLAG          FUNCTION      TEST PARAMETERS */
{"system.hostname", 0,          SYSTEM_HOSTNAME,  NULL};
```

在这里就将 “`system.hostname`” 和方法 “`SYSTEM_HOSTNAME`” 对应起来了。

20.2 system.cpu.load

通过上一节大家已经对于 Zabbix 内置监控项的实现了解了个大概。我们这一节再看一下 `system.cpu.load` 的实现，相比 `system.hostname`，它更加能体现 Zabbix 对于内置监控项的设计思路，

因为 `system.cpu.load` 包括了对于内置监控项参数的解析等功能。

关于 CPU 的内置监控，都在 `cpu.c` 中，它一共定义了以下 5 个方法。

- (1) `SYSTEM_CPU_NUM` : CPU 的个数。
- (2) `SYSTEM_CPU_UTIL` : CPU 的各个状态数据，比如 `iowait`，支持针对某个 CPU。
- (3) `SYSTEM_CPU_LOAD` : CPU 负载。
- (4) `SYSTEM_CPU_SWITCHES` : CPU 上下文切换数量。
- (5) `SYSTEM_CPU_INTR` : CPU 中断。

获取监控项的参数使用的是“`get_rparam`”方法，它接受两个参数：第一个是对于一个监控项的对象 `request`，第二个对象是参数的位置。比如“`foo[a,b,]`”，使用 `get_rparam(request, 1)` 就获取到了“a”。获得参数后，再使用 `strcmp` 方法来判断是否有需要的参数。比如 `system.cpu.load` 有固定接收的几个参数（`avg1`，`avg5` 等）。这是 C 语言中比较字符串是否相等的方法，返回值为 0 即表示相等。

下面我们看 `cpu.c` 中计算 CPU 负载的代码。

```
int SYSTEM_CPU_LOAD (AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char      *tmp;
    int mode, per_cpu = 1, cpu_num;
    double    load[ZBX_AVG_COUNT], value;
    if (2 < request->nparam)
        return SYSINFO_RET_FAIL;
    tmp = get_rparam(request, 0);
    if (NULL == tmp || '\0' == *tmp || 0 == strcmp(tmp, "all"))
        per_cpu = 0;
    else if (0 != strcmp(tmp, "percpu"))
        return SYSINFO_RET_FAIL;
    tmp = get_rparam(request, 1);
    if (NULL == tmp || '\0' == *tmp || 0 == strcmp(tmp, "avg1"))
        mode = ZBX_AVG1;
    else if (0 == strcmp(tmp, "avg5"))
        mode = ZBX_AVG5;
```

```

else if (0 == strcmp(tmp, "avg15"))
    mode = ZBX_AVG15;
else
    return SYSINFO_RET_FAIL;
if (mode >= getloadavg(load, 3))
    return SYSINFO_RET_FAIL;
value = load[mode];
if (1 == per_cpu)
{
    if (0 >= (cpu_num = sysconf(_SC_NPROCESSORS_ONLN)))
        return SYSINFO_RET_FAIL;
    value /= cpu_num;
}
SET_DBL_RESULT(result, value);
return SYSINFO_RET_OK;
}

```

核心代码是使用 `getloadavg` 方法获取一个当前 CPU 负载的数组——`load`，然后根据输入的参数，从 `load` 数组中取出需要的数值。

这里好像没有像 `hostname.c` 中那样写明这个方法在前端是使用“`system.hostname`”调用的。是的，`cpu.c` 和前端 key 的对应关系，不是写在 `cpu.c` 里的，而是写在 `linux.c` 中，代码如下。

```

ZBX_METRIC      parameters_specific[] =
/* KEY FLAG FUNCTION TEST PARAMETERS */
{
    {"kernel.maxfiles", 0, KERNEL_MAXFILES, NULL},
    {"kernel.maxproc", 0, KERNEL_MAXPROC, NULL},
    ...
}

```

CPU 负载的 key 也在里面，如下。

```

{"system.cpu.load", CF_HAVEPARAMS, SYSTEM_CPU_LOAD, "all,avg1"},

```

里面除了 key、调用的方法，最后一列表示的是什么参数都不输入的默认参数。

第 25 章

经典案例



第五部分 社区和开源

★ 第 21 章 典型案例分析

★ 第 22 章 Zabbix 代码问题和解决

★ 第 23 章 PPTV 的 Zabbix 监控体系

★ 第 24 章 Zatree

★ 第 25 章 Zabbix 第三方插件

★ 第 26 章 微信公众平台报警

★ 第 27 章 社区论坛

第 21 章

典型案例分析

在使用 Zabbix 过程中，会碰到各种各样的问题，有的容易解决，有的却非常困难。在这一章中，笔者会把一些碰到的非常困难的问题的解决方法分享给各位。

在笔者使用 Zabbix 和各种开源产品的过程中，和很多朋友都有过交流，发现大家对于开源产品出的问题，经常束手无策，把它们当做一个黑盒。笔者认为，要用好一个产品，一定要从代码级别去分析解决问题。很多朋友总觉得开源工具是一个非常高深的东西，它的代码一定非常难懂，很难从其中找到解决问题的方法。其实不然，开源工具也是工程师开发的，它也有很多问题，它的代码更不是“不可能读懂的”。应该从代码级别去分析问题，并且解决问题，使得自己对于开源工具的理解能够更深，从而提升自己的水平。

在本章中，每一节会分析一个问题，大多数问题都会从代码上进行分析，希望大家能从中找到感觉，并在工作中更多的使用这种方法排查问题。

21.1 前端显示Zabbix server停止工作问题

有很多朋友碰到过一个问题，它的现象就是明明 Zabbix Server 和 Zabbix 的前端都运行很正常，但在前端会有这个提示，而且，还显示 Zabbix 没有运行，如图 21-1 所示。这是个很典型的问题，我们想去看是什么问题引起的告警，又无从下手。这一节大家就跟着笔者一起抽丝剥茧，把这个问题搞清楚，并将其解决吧。

Zabbix server is not running: the information displayed may not be current.

Parameter	Value
Zabbix server is running	No

图21-1

现有的信息就是这条报警，先看看这个报警是在 PHP 代码中的什么地方。我们发现除了 locale 文件（即各种语言的翻译文件外），只有在 jsrpc.php 中出现过，又根据 PHP 文件名，可以猜测这是一个提供 RPC 服务的接口 PHP，下面是代码片段。

```
case 'zabbix.status':
    $session = Z::getInstance() -> getSession();
    if (!isset($session['serverCheckResult']) || ($session['serverCheckTime']
+ SERVER_CHECK_INTERVAL) <= time()) {
        $zabbixServer = new CZabbixServer($ZBX_SERVER, $ZBX_SERVER_PORT, ZBX_
SOCKET_TIMEOUT, 0);
        $session['serverCheckResult'] = $zabbixServer->isRunning();
        $session['serverCheckTime'] = time();
    }
    $result = array(
        'result' => (bool) $session['serverCheckResult'],
        'message' => $session['serverCheckResult'] ? '' : _('Zabbix server is
not running: the information displayed may not be current.')
    );
    break;
```

从代码分析，发现是否显示“Zabbix server is not running”，是根据 \$session['serverCheckResult'] 来判断的，而这个变量又是从 \$zabbixServer 的 isRunning 方法获取的。isRunning 这个方法的代码在 include/classes/server/CZabbixServer.php 中，片段如下。

```
/**
 * Returns true if the Zabbix server is running and false otherwise.
 *
 * @return bool
 */
```

```

public function isRunning() {
    return (bool) $this->connect();
}

```

从注释来说，方向是对的，这个方法是用来判断 Zabbix server 是否在运行的。看代码它是使用 this 指针，那 connect 方法一定在本 PHP 文件中了，代码如下。

```

/**
 * Opens a socket to the Zabbix server. Returns the socket resource
 * if the connection has been established or
 * false otherwise.
 *
 * @return bool|resource
 */
protected function connect() {
    if (!$this->socket) {
        if (!$this->host || !$this->port) {
            return false;
        }
        if (!$socket = @fsockopen($this->host, $this->port, $errorCode,
            $errorMsg, $this->timeout)) {
            switch ($errorMsg) {
                case 'Connection refused':
                    $dErrorMsg = _s("Connection to Zabbix server
                    \"%s\" refused. Possible reasons:\n1. Incorrect server
                    IP/DNS in the \"zabbix.conf.php\";\n2. Security
                    environment (for example, SELinux) is blocking the
                    connection;\n3. Zabbix server daemon not running;\n4.
                    Firewall is blocking TCP connection.\n", $this->host);
                    break;
                case 'No route to host':
                    $dErrorMsg = _s("Zabbix server \"%s\" can not
                    be reached. Possible reasons:\n1. Incorrect server
                    IP/DNS in the \"zabbix.conf.php\";\n2. Incorrect
                    network configuration.\n", $this->host);

```

```

        break;
    case 'Connection timed out' :
        $dErrorMsg = _s("Connection to Zabbix server
        \" %s\" timed out. Possible reasons:\n1. Incorrect server
        IP/DNS in the \" zabbix.conf.php\" ;\n2. Firewall is
        blocking TCP connection.\n" , $this->host);
        break;
    case 'php_network_getaddresses: getaddrinfo failed:
    Name or service not known' :
        $dErrorMsg = _s("Connection to Zabbix server
        \" %s\" failed. Possible reasons:\n1. Incorrect server
        IP/DNS in the \" zabbix.conf.php\" ;\n2. Incorrect DNS
        server configuration.\n" , $this->host);
        break;
    default:
        $dErrorMsg = '' ;
    }
    $this->error = $dErrorMsg.$errorMsg;
}
$this->socket = $socket;
}
return $this->socket;
}

```

从方法的名称（connect）和方法的注释，大家应该已经明白 connect 方法的作用就是和 Zabbix server 建立一个网络连接了。连接建立的代码如下。

```
@fsockopen($this->host, $this->port, $errorCode, $errorMsg, $this->timeout)
```

那么 \$this->host 和 \$this->port 是哪里定义的呢？从 this 指针看出，肯定就在本 PHP 文件中。一般这种变量定义都在文件的开头处，我们从上往下看，很快就在这个类的构造函数中发现，这两个变量，是在初始化这个类的过程中传入的参数。CZabbixServer 的构造函数如下。

```

/**
 * Class constructor.
 *

```



```

* @param string $host
* @param int $port
* @param int $timeout
* @param int $totalBytesLimit
*/
public function __construct($host, $port, $timeout, $totalBytesLimit)
{
    $this->host = $host;
    $this->port = $port;
    $this->timeout = $timeout;
    $this->totalBytesLimit = $totalBytesLimit;
}

```

回过头去看最早的 jsrpc.php，它是这么初始化 CZabbixServer 的：

```

$zabbixServer = new CZabbixServer($ZBX_SERVER, $ZBX_SERVER_PORT, ZBX_
SOCKET_TIMEOUT, 0);

```

关键就是 \$ZBX_SERVER 和 \$ZBX_SERVER_PORT，这两个参数在 jsrpc.php 中没有再出现。这两个变量是全部大写的，判断可能是全局变量，然后使用 grep 把所有使用 \$ZBX_SERVER 的地方找出来，发现这个变量是在 conf/zabbix.conf.php 中设置的，代码如下。

```

<?php
// Zabbix GUI configuration file
global $DB;
$DB["TYPE"] = 'MYSQL';
$DB["SERVER"] = 'localhost';
$DB["PORT"] = '0';
$DB["DATABASE"] = 'zabbix';
$DB["USER"] = 'zabbix';
$DB["PASSWORD"] = 'zabbix_password';
// SCHEMA is relevant only for IBM_DB2 database
$DB["SCHEMA"] = '';
$ZBX_SERVER = 'localhost';
$ZBX_SERVER_PORT = '10051';
$ZBX_SERVER_NAME = '';

```

```
$IMAGE_FORMAT_DEFAULT = IMAGE_FORMAT_PNG;
```

```
?>
```

好了，整个分析过程结束了。我们来整理下思路：“Zabbix server is not running”的出现，是因为 Zabbix 前端无法根据配置中的 Zabbix server 的 IP 和端口与 Zabbix server 建立连接造成的。参数 \$ZBX_SERVER 和 \$ZBX_SERVER_PORT 是在前端的配置过程中设置的，在更改了 Zabbix server 的 IP 或者端口后，并不会再去运行一次 PHP 中的 installation 过程，所以会出现问题。解决的方法就是将这个参数改成正确的 Zabbix server IP 和端口就行了。

提示问题解决后，Zabbix 也会显示为正常，如图 21-2 所示。




图21-2

继续研究，看看这里的“Yes”和“No”是怎么计算出来的。首先直接看 PHP 文件 report1.php，发现构造这一行的代码如下。

```
$reportWidget->addItem(make_status_of_zbx());
```

找到 make_status_of_zbx 方法的定义在 include/blocks.inc.php 中，显示“Yes”和“No”的逻辑如下。

```
$table->addRow(array(
    _('Zabbix server is running'),
    new CSpan($status['zabbix_server'], ($status['zabbix_server'] == _
('Yes') ? 'off' : 'on')),
    isset($ZBX_SERVER, $ZBX_SERVER_PORT) ? $ZBX_SERVER.':'.$ZBX_SERVER_
PORT : _('Zabbix server IP or port is not set!')
));
```

我们发现是根据 \$status['zabbix_server'] 来判断的，而 \$status 是通过 get_status 方法获取的。继续看 get_status 方法，发现它是在 include/func.inc.php 中定义的，和 Zabbix server 相关的逻辑如下。

```
// server
$zabbixServer = new CZabbixServer($ZBX_SERVER, $ZBX_SERVER_PORT, ZBX_
SOCKET_TIMEOUT, 0);
$status['zabbix_server'] = $zabbixServer->isRunning() ? _('Yes') : _
```

```
('No');
```

也是通过 `$zabbixServer` 的 `isRunning` 方法来获取状态的。所以，当解决了提示的问题后，这个表格中的 Zabbix server 的状态也恢复正常了。

问题解决了，可能有的读者有疑问：为什么在配置文件中把 Zabbix Server 的 IP 或者端口号都写错了，前端还是运行正常呢。其实 Zabbix 前端的数据以及对数据的修改，都是直接和 Zabbix 数据库通信的，和 Zabbix Server 没有直接的关系。

21.2 Item设置了但没有数据

这个问题是最常见的了，“为什么我配置的 Item 没有数据啊？”可以像 Windows 的帮助那样，一步一步地来排查问题。比如打印机无法使用，Windows “帮助” 会这样做：

(1) 您的打印机连接上电源，电源指示灯是亮着的吗？

(2) 电脑识别您的打印机了吗？

(3) 换一个打印机驱动试试。

从基础的问题开始排查起，直到解决。下面我们会这种思路来一起解决难题。

21.2.1 看页面是否有报错

在“Configuration”→“Host”中单击出问题的 Host，找到出问题的 Item，看看在 Error 列是否有一个红色的叉，如果有，把鼠标移上去看看提示信息，如图 21-3 所示。



Type	Applications	Status	Error
Zabbix agent		Not supported	
Received value [Zabbix server] is not suitable for value type [Numeric (unsigned)] and data type [Decimal]			
Zabbix agent		OK	

图21-3

看起来是 Item 的返回值的类型和在 Item 中设置的类型不匹配。在 `zabbix_server.log` 中也会记录下这个问题：

```
item [HostABC:agent.hostname] became not supported: Received value [Zabbix
server] is not suitable for value type [Numeric (unsigned)] and data type
[Decimal]
```

这非常常见。在使用 Zabbix 1.8.8 的时候, 如果类型错误, 页面上是不会告诉我们的, 只会显示 “Not supported”, 需要看 zabbix_agent 的 debug 日志才行。这个问题的解决办法就是根据 Item 的返回值 (提示文字中方括号中的 “Zabbix server”), 来选择正确的 Item 配置。

21.2.2 Zabbix Server和Zabbix Agent的网络是否互通

如果在页面上根本没有报错的时候, 就要从系统入手了, 首先要确定的是 Zabbix Server 和 Zabbix Agent 在网络上是否互通的, 检查方法很简单, 就是 telnet, 代码如下。

```
[root@zabbix_agent /]$ telnet zabbix_server 10051
Trying 127.0.0.1...
Connected to zabbix_server (127.0.0.1).
Escape character is '^]'.
[root@zabbix_server /]$ telnet zabbix_agent 10050
Trying 127.0.0.1...
Connected to zabbix_agent (127.0.0.1).
Escape character is '^]'.

```

具体的端口如果有过变动, 那就要相应调整命令。上面的情况说明网络端口是通的。如果 telnet 通了以后马上被关闭, 形如:

```
[root@host /]$ telnet zabbix_agent 10050
Trying 127.0.0.1...
Connected to zabbix_agent (127.0.0.1).
Escape character is '^]'.
Connection closed by foreign host.

```

那就是 zabbix_agentd.conf 中 “Server” 参数的问题了。需要将它配置为 Zabbix server 的 IP。注意, 如果 Zabbix Server 和 Zabbix Agent 的通信是经过外网的, “Server” 还需要配置 Zabbix Server 的出口 IP。因为经过了外网, Zabbix Agent 看到的 IP 可能已经不是 Zabbix Server 自己网卡上的 IP 了。对于这种问题, 要知道 Zabbix Server 对于其他外网机器的可见 IP, 可以使用 ifconfig.me 网站的服务, 具体如下。

```
[root@zabbix_server /]$ curl ifconfig.me
121.32.130.XXX

```

然后把这个 IP 写到 Zabbix Agent 的“Server”中。

在保证了两边 telnet 都通了以后，进行下一步。

21.2.3 zabbix_get是否能够获取到数据

假设架构是最简单的 Zabbix Server 和 Zabbix Agent，在排查了网络问题后，在这一小节使用 Zabbix 自带的 zabbix_get 来测试 Item 的数据是否获取正常。

zabbix_get 和 zabbix_get 在 Zabbix 安装目录的 bin 文件夹下，用来获取某台 Host 的某个 key 的数据，它接受以下 6 个参数。

- (1) -s 或者 --host：后面跟 hostname 或者目标 Host 的 ip，必填。
- (2) -p 或者 --port：目标 Host 的 Zabbix Agent 端口，默认是 10050。
- (3) -I 或者 --source-address：发起这次 zabbix_get 的起始 IP。
- (4) -k 或者 --key：想要获取数据的 key。
- (5) -h 或者 --help：显示帮助信息。
- (6) -V 或者 --version：输出版本信息。

我们可以这样测试，比如想要获取的是 foo.echo 这个 key，就使用下面的命令。

```
zabbix_get -s zabbix_agent -k foo.echo
```

看有没有返回值。

使用 zabbix_get 这一步的目的是：判断 Item 执行的脚本是否成功。很多时候，Item 没有数据是因为监控脚本写错了，使用 zabbix_get 就可以排除这一点。

21.2.4 总结

经过上面几步，基本上 90% 的问题就能解决了。如果还不能解决问题，比如说 zabbix_get 也没有数据，那就要到 Zabbix Agent 服务器上去检查是不是脚本有问题了。对于这样常见的问题，大家一定要理清思路，一层一层地排查，最好不要一有问题就到论坛或者 QQ 群去问，在问之前先思考一下，才会不断掌握解决问题的方法，从而真正得到提高。

21.3 一个扫描history全表的SQL问题

这个问题的表现就是 Zabbix 后端的 MySQL 发现一个查询在扫描 history 全表,SQL 语句如下。

```
select x, x from history_uint where itemid=12345 and clock<= now();
```

第一反应就是 Graph 在扫描全表,因为除了这个地方,我们想不出还有什么地方需要扫描整个 history 表。

第一反应是怀疑是不是 itemid 为 12345 的 Item 没有把数据存储到 trends 表,而是直接存储到了 history,比如如图 21-4 所示的设置。

History storage period (in days)	<input type="text" value="90"/>
Trend storage period (in days)	<input type="text" value="0"/>

图21-4

但检查了 Item,发现它设置存储了 Trend,所以上面这种怀疑排除。

看来只能从代码去找这条 SQL 的发生原因了,下面是 DBA 贴出的完整 SQL。

```
10731049 Query begin
10731049 Query select i.itemid,i.status,i.lastclock,i.prevorgvalue,i.
delta,i.multiplier,i.formula,i.history,i.trends,i.lastns,i.hostid,i.
inventory_lin
k,hi.inventory_mode,i.valuemapid,i.units,i.error from items i left join
host_inventory hi on hi.hostid=i.hostid where status in (0,3) and i.itemid
in (120170,126770,12
8900,131810,132890,145430,148250) order by i.itemid
10731049 Query update items set lastclock=1392778610,lastns=9228110
00,prevorgvalue='1904695255',prevvalue=lastvalue,lastvalue='1523' where
itemid=1201
70;
10731049 Query update items set lastclock=1392778610,lastns=903523000,p
revorgvalue='0',prevvalue=lastvalue,lastvalue='0' where itemid=126770;
10731049 Query update items set lastclock=1392778610,lastns=878256000,p
revvalue=lastvalue,lastvalue='777' where itemid=128900;
10731049 Query update items set lastclock=1392778610,lastns=878234000,p
```

```

revvalue=lastvalue,lastvalue='528' where itemid=131810;
10731049 Query update items set lastclock=1392778610,lastns=9263020
00,prevorgvalue='3447327737',prevvalue=lastvalue,lastvalue='1024' where
itemid=1328
90;
10731049 Query update items set lastclock=1392778610,lastns=9057340
00,prevorgvalue='2370646676',prevvalue=lastvalue,lastvalue='3684' where
itemid=1454
30;
10731049 Query update items set lastclock=1392778610,lastns=923759000,p
revvalue=lastvalue,lastvalue='100.000000' where itemid=148250
10731049 Query insert into history (itemid,clock,value,ns) values (14825
0,1392778610,100.000000,923759000);
10731049 Query insert into history_uint (itemid,clock,value,ns) values
(131810,1392778610,528,878234000),(128900,1392778610,777,878256000),
(126770,139
2778610,0,903523000),(145430,1392778610,3684,905734000),(120170,139277
8610,1523,922811000),(132890,1392778610,1024,926302000)
10731049 Query select i.itemid,i.key_,h.host,i.type,i.history,i.
lastvalue,i.prevvalue,i.hostid,i.value_type,i.delta,i.prevorgvalue,i.
lastclock,i.units
,i.multiplier,i.formula,i.status,i.valuemapid,i.trends,i.data_type,h.
status from hosts h,items i where i.hostid=h.hostid and i.itemid in
10731049 Query select value from history_uint where itemid=128900 and
clock>1392774950 and clock<=1392778550 order by clock desc limit 1
10731049 Query select value from history_uint where itemid=132890 and
clock<=1392778610
10731049 Query commit

```

出问题的SQL是倒数第二句，即“10731049 Query select value from history_uint where itemid=132890 and clock<=1392778610”。然后怀疑这个SQL不是Zabbix Server跑的，经过确认，是Zabbix Server运行的，如图21-5所示。


```

# Time: 140219 19:22:58
# User@Host: zabbix[zabbix] @ [10.200.100.33] Id: 10731016
# Query_time: 2.399352 Lock_time: 0.000087 Rows_sent: 24692 Rows_examined: 24692
SET timestamp=1392808978;
select value from history_uint where itemid=122181 and clock<=1392808222;
# User@Host: zabbix[zabbix] @ [10.200.100.33] Id: 10731017
# Query_time: 0.316634 Lock_time: 0.000190 Rows_sent: 24712 Rows_examined: 24712
SET timestamp=1392808978;
select value from history_uint where itemid=41959 and clock<=1392807079;
# User@Host: zabbix[zabbix] @ [10.200.100.33] Id: 10731043
# Query_time: 6.989542 Lock_time: 0.000103 Rows_sent: 24717 Rows_examined: 24717
SET timestamp=1392808978;
select value from history_uint where itemid=120296 and clock<=1392808258;
# User@Host: zabbix[zabbix] @ [10.200.100.33] Id: 10731053
# Query_time: 0.979413 Lock_time: 0.000140 Rows_sent: 24712 Rows_examined: 24712
SET timestamp=1392808978;
select value from history_uint where itemid=101289 and clock<=1392808329;
# User@Host: zabbix[zabbix] @ [10.200.100.33] Id: 10731056
# Query_time: 0.635522 Lock_time: 0.000183 Rows_sent: 18473 Rows_examined: 18473
SET timestamp=1392808978;
select value from history_uint where itemid=132015 and clock<=1392808815;

```

图21-5

接着找 SQL 的运行规律，发现这个 SQL 一直在运行，而且造成 MySQL 的磁盘 I/O 接近 100%，Zabbix 截图如图 21-6 所示。

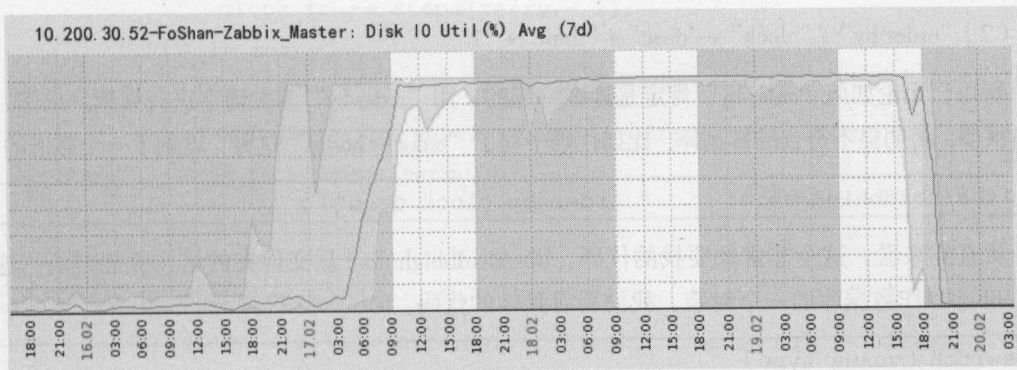


图21-6

先分析下整个事务里面的 SQL，它可以分为以下几类。

(1) `select i.itemid,i.status,i.lastclock,i.prevorgvalue,i.delta,i.multiplier,i.formula,i.history,i.trends,i.lastns,i.hostid,i.inventory_link,hi.inventory_mode,i.valuemapid,i.units,i.error` : 很明显，这个是在查询 Items 的一些信息。

(2) `update items set lastclock=XXX,lastns=XXX,lastvalue=XXX where itemid=XXX` : 这个 SQL 是在更新 Items 表的记录，更新的内容是时间点和获取到的监控值。

(3) `insert into history (itemid,clock,value,ns) values (XXX,XXX,XXX,XXX) where itemid=XXX` : 这个 SQL 是将 Item 的数据插入到 history 表中。

(4) select value from history where itemid=XXX and clock between XXX and XXX：这是从 history 表中查询数据，也就是出问题的 SQL 的位置。

确认使用的是 Zabbix 2.0 后，开始从代码入手来找问题。首先看出问题的 SQL：

```
select value from history_uint where itemid=132890 and clock<=1392778610
```

我们很容易将它和 (4) 归为一类，都是从 history 中获取数据。那么，是什么地方在调用这两条 SQL 呢？在拼接 SQL 的时候，条件都是使用变量的，比如“itemid=XXX”会写成“itemid=+ITEM_ID”这种形式，所以笔者决定使用“order by clock desc limit 1”在 Zabbix 代码中搜索，但是没有找到，说明其中肯定有变量。根据分析，在“order by clock desc limit 1”中有可能是下面这几种情况：

(1) “order by clock desc”+“limit”+“1”

(2) “order by”+“clock”+“desc”+“limit”+“1”

所以，无论如何“order by”肯定是连在一起的，用“order by”为关键字进行搜索。果然，搜索出很多，将明显不符合的排除掉，比如代码中写了“order by hostid”这种，就剩下一个靠谱的了：

```
libs/zbxdbhigh/db.c:      " order by clock desc");
```

从直觉来看，这个非常像要找的代码，libs/zbxdbhigh/db.c 从文件夹的结构就能判断，这个是 Zabbix 操作数据库的一个封装，我们看看具体的代码。

```
switch (value_type)
{
    case ITEM_VALUE_TYPE_FLOAT:
    case ITEM_VALUE_TYPE_UINT64:
    case ITEM_VALUE_TYPE_STR:
        offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " order
by clock desc");
        break;
    default:
        offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " order
by id desc");
        break;
}
```

从代码看，是在拼接字符串。这段代码是属于 DBget_history 方法的，一看名字就知道，是从数据库查询 history 名字的，和之前的判断是吻合的。看来这段错误的 SQL，就是从这里来的。出问题的 SQL 使用了“clock<=”的条件，相比“看上去”正常的它上面那条 SQL，它少了“clock>”这个条件，我们接下来看看 db.c 拼接这些条件的代码。

```

if (NULL == ts)
{
    if (0 != last_n && 0 == clock_from && 0 != clock_to)
    {
        const int steps[] = {SEC_PER_HOUR, SEC_PER_DAY, SEC_PER_WEEK, SEC_
PER_MONTH};
        if (0 != retry)
            clock_to -= steps[retry - 1];
        if (4 != retry)
        {
            offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, "
and clock>%d", clock_to - steps[retry]);
        }
    }
    if (0 != clock_from)
        offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " and
clock>%d", clock_from);
    if (0 != clock_to)
        offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " and
clock<=%d", clock_to);
}
else if (1 == retry)
{
    offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " and
clock=%d", sec);
    if (-1 != ns)
        offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " and
ns<%d", ns);
}

```

```

else
    offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " and
clock=%d and ns=%d", ts->sec, ts->ns);

```

可以看见, 如果传递给 DBget_history 的参数中, “clock_from” 的值为 “0”, 那么就不会有 “clock>” 这个限制条件。

再看 db.c 中的一段代码。

```

switch (function)
{
    case ZBX_DB_GET_HIST_MIN:
    case ZBX_DB_GET_HIST_AVG:
    case ZBX_DB_GET_HIST_MAX:
    case ZBX_DB_GET_HIST_SUM:
        offset = zbx_snprintf(sql, sizeof(sql), "select %s(%s)", func
[function], field_name);
        break;
    case ZBX_DB_GET_HIST_COUNT:
        offset = zbx_snprintf(sql, sizeof(sql), "select %s(*)", func
[function]);
        break;
    case ZBX_DB_GET_HIST_DELTA:
        offset = zbx_snprintf(sql, sizeof(sql), "select max(%s)-min(%s)",
field_name, field_name);
        break;
    case ZBX_DB_GET_HIST_VALUE:
        offset = zbx_snprintf(sql, sizeof(sql), "select %s", field_
name);
        break;
    default:
        assert(0);
}

```

结合出问题 SQL 的 “select value”, 可以判断: 传入 DBget_history 的 function 参数的值是 “ZBX_DB_GET_HIST_VALUE”。

大家不要忽略了,出问题的 SQL 中,有一个限制,即“limit 1”,这是什么地方拼接的 SQL 呢? 仔细看看 DBget_history 这个方法,找到了这样一条代码:

```
result = DBselectN(sql, last_n - h_num);
```

看看 DBselectN 的代码,发现它是调用了“rc = zbx_db_select_n(query, n);”,果然,在 libs/zbxdb/db.c 中,发现 zbx_db_select_n 就是拼接 limit 的地方,代码如下。

```
#elif defined(HAVE_MYSQL)
return zbx_db_select("%s limit %d", query, n);
```

再看调用 DBselectN 的前提条件,发现如果 SQL 中有“limit”,那么“last_n”不能为“0”,证据在“if(0 != last_n)”。而出问题的 SQL 中,恰恰没有“limit”这个限制条件,和它类似功能在同一个事务中执行的 SQL,却有“limit”,说明,执行出问题 SQL 的时候,“last_n”的值为“0”。

接着看,是什么地方调用了 DBget_history。

```
$ grep -rl 'DBget_history' *
libs/zbxdbhigh/db.c
libs/zbxserver/evalfunc.c
libs/zbxserver/expression.c
zabbix_server/poller/checks_aggregate.c
```

可以把 db.c 先排除,evalfunc.c 是计算 function 的地方(“eval”是“evaluate”的意思,译为“评估”),function 和 Trigger 有关。看这 4 个文件,db.c 已经排除了,因为这是定义 DBget_history 的地方,evalfunction 和 expression 是一个功能,所以它们是一组。

再看看 checks_aggregate.c, DBget_history 是在 evaluate_aggregate 方法中调用的,aggregate 即为 Item 的一种计算方式。aggregate 类型的 Items,是把多个 Items 的结果聚合起来的一个 Item。比如监控了 5 个 IDC 的网络出口流量,那么就需要一个 Item 将这 5 个 Item 的值聚合起来,表示为网络出口总流量。那我们的 SQL 会不会是这里出现的呢? 答案是“否”,因为 checks_aggregate.c 中出现 DBget_history 的代码如下。

```
while (NULL != (row = DBfetch(result)))
{
    ZBX_STR2UINT64(itemid, row[0]);
    value_type = (unsigned char)atoi(row[1]);
    h_value = DBget_history(itemid, value_type, item_func, clock_from, 0,
        NULL, NULL, 0);
```

```

    if (NULL != h_value[0])
        evaluate_one(item, &value, &num, grp_func, h_value[0], value_type);
    DBfree_history(h_value);
}

```

其中，有一个 while 循环，再结合 aggregate item 的特性，可以判断，这里在做的事情，是将多个 Item 的 history 值查出来，然后汇总出 aggregate item 的值。笔者把这段代码翻译成如下代码。

```

while:
value = DBget_history()
if value != NULL:
    aggregate_value += value
    update_history(aggregate_value)

```

所以，如果出问题的 SQL 是在这里出现的，那么在事务的最后，一定会有一步是更新 aggregate 的操作，但是事务中没有这样的 SQL，所以排除 checks_aggregate.c 的嫌疑。

看来嫌疑已经锁定在 expression.c 和 evalfunc.c 中了，由于 function 是属于 expression 的，所以先从 expression 开始看。expression.c 的代码的主要工作就是在 DBget_history 上又封装了一层，比如类似“DBget_item_value”的方法，并没有实际操作。

在看 evalufunc.c 之前，先回顾下 DBget_history 的定义。

```

char      **DBget_history(zbx_uint64_t itemid, unsigned char value_type,
int function, int clock_from, int clock_to,
        zbx_timespec_t *ts, const char *field_name, int last_n)

```

从出问题的 SQL 我们看得出，SQL 只有“clock<=”而没有“clock>”，所以，在调用 DBget_history 的时候，“clock_from”为“0”，而“clock_to”不为“0”，现在要在 evalfunc.c 中寻找这样的调用，结果如下。

(1) DBget_history(item->itemid, item->value_type, ZBX_DB_GET_HIST_COUNT, now - arg1, now, NULL, NULL, 0);

(2) DBget_history(item->itemid, item->value_type, ZBX_DB_GET_HIST_VALUE, 0, now, NULL, NULL, arg1);

(3) DBget_history(item->itemid, item->value_type, ZBX_DB_GET_HIST_VALUE, now - arg1,

```
now, NULL, NULL, 0);
```

```
(4) DBget_history ( item->itemid, item->value_type, ZBX_DB_GET_HIST_SUM, now - arg1,
now, NULL, NULL, 0);
```

还有一些代码类似就不一一贴出来了。不同在于以下几点。

(1) 传给 DBget_history 的方法不同,即上面代码中的“ZBX_DB_GET_HIST_VALUE”和“ZBX_DB_GET_HIST_SUM”。

(2) clock_from 和 clock_to 的组合一共有两种:一种是“clock_from=now-arg1, clock_to=now”,一种是“clock_from=0, clock_to=now”。

从出问题的 SQL 可以看出,显然我们是“ZBX_FLAG_VALUES”的,因为我们有一个又由于 DBget_history 的“last_n”参数是“0”,所以 arg1 就是“0”。再结合之前已经判断出, function 的参数是“ZBX_DB_GET_HIST_VALUE”,那么就很容易找代码了。

到此为止,我们整理下,出问题的 SQL 调用 DBget_history 的参数如下:

(1) clock_from=0

(2) last_n = 0, 即 arg1 = 0

(3) field_name=NULL

从看到的代码来看,有可能出现问题的 SQL 只可能是:

```
DBget_history (item->itemid, item->value_type, ZBX_DB_GET_HIST_VALUE, 0,
now, NULL, NULL, arg1)
```

那么 flag 就肯定是“ZBX_FLAG_VALUES”。并且其中 arg1=0。只有这样,才会出现这种问题。所以在配置 trigger 的时候,使用了“#0”这种写法。

在设置 function 中的时间范围时有两种方式:一种以“#”开头,表示次数的:一种不以“#”开头,表示时间。在拼接 SQL 的时候,这两个是要区别对待的。在 evalfunc.c 中,“get_function_parameter_unit”方法就是做这个事情的,当以“#”开头时,flag 会被置为“ZBX_FLAG_VALUES”,否则,会被置为“ZBX_FLAG_SEC”。而 arg1、args2...就是 function 中解析出的参数。

我们再反过来理一下思路, function 参数为“#0”时候,代码中逻辑是怎样的。

(1) 由于 function 是 ZBX_DB_GET_HIST_VALUE, 且 field_name=NULL, 所以 SQL 以 “select value” 开头, 加上从 history 取数据 (history_unit 类似), 那么 SQL 就是 “select value from history where itemid=×××”

(2) 因为 last_n=0, clock_from=0, clock_to=now, 所以会加上 clock<= 的限制, 则目前的 SQL 为 “select * from history where itemid=××× and clock <= ×××”

这就是我们出问题的 SQL 的由来。

最后一查, 果然如此, 如图 24-4 所示。

Name	{HOST.NAME} {#IFNAME} Network Traffic (in) mo	
Expression	{Template_OS_Linux_comm:net.if.in[{#IFNAME}].last(#0)}>838041600	Add

图24-4

总结下, Zabbix 对于 “#”, 表示 “第 × 次” 的数据, 显然 “第 0 次” 的数据是没有意义的, 从代码来看, 如果用的是 “#1”, 那么 last_n 就不等于 0, 从而就会加上 clock>××× 的逻辑, 代码如下。

```

if (0 != last_n && 0 == clock_from && 0 != clock_to)
{
    const int steps[] = {SEC_PER_HOUR, SEC_PER_DAY, SEC_PER_WEEK, SEC_PER_MONTH};
    if (0 != retry)
        clock_to -= steps[retry - 1];
    if (4 != retry)
    {
        offset += zbx_snprintf(sql + offset, sizeof(sql) - offset, " and
clock>%d", clock_to - steps[retry]);
    }
}

```

由于错误的输入了 #0, 所以才造成这样的问题。

后来我发现 Zabbix 2.2 已经修复了这个 bug, 在 evalfunc.c 中有非常大的改动, 在 libs/zbxdbhigh/db.c 中已经没有 DBget_history 方法了, 使用最新版本 Zabbix 2.2 的读者不用担心。

这个问题，算是本书中最复杂的一个从代码级别分析、解决问题的例子了，虽然说到 Zabbix 2.2 中已经用不到了，但排查的思路还是值得分享的。

21.4 解决问题的思路

在和网上朋友交流 Zabbix 的时候，我发现大家对一个问题，没有一个很清晰的思路，或者说懒得去刨根问底解决这个问题，而是希望去通过“问”的方式，快速获取答案。我觉得一些自己能力以内的问题，完全可以调动自己的潜力去解决，这样才能带来技术上的不断进步。

从我自己的经验来说，之前在 PPTV 做 Zabbix 和现在在唯品会做数据科学的基础架构，都是之前没有接触过的，并且国内做的人还不多。对于碰到的问题如何去研究呢？我分享下我的经验，先要从问题本身收集详细的数据，最重要的就是日志。如果要报错日志，是最简单的，先 Google，如果有答案最好，如果没有。那么直接从源代码入手，grep 找到抛出这个错误的代码，分析出现这个问题的可能。

另一点我想谈的是，很多时候我们的问题是这样的——一个好好运行的系统，突然出问题了，而出问题的原因却找不到。针对这种诡异的问题，只要记住一点，所有系统的异常都是由变化引起的。可能是外部环境，可能是服务器本身。从“变”去分析系统异常的原因。

从上一段的“找变化”总结一下，我考虑问题的思路是“横向比较，纵向比较”。这个说起来很简单，但真的能把这个思想融会贯通、在处理问题中真正使用的，不多。我举个例子，我一个集群中的一台节点突然出问题了。这时，我们先横向比较——这个节点和集群中的其他节点是否有不同；再纵向比较——这个节点现在和之前比较，有什么不同，为什么现在要出问题，而不是一小时前。这样就能沿着正确的方向解决问题了

第 22 章

Zabbix代码问题和解决

笔者在使用 Zabbix 1.8 的过程中，一共提交了三个 Patch，都是和性能相关的。一个是 nextid 问题，一个是数据库拼接 SQL 问题，还有一个是代码的 bug——一个 connection 使用后没有关闭。虽然现在已经是 Zabbix 2.2 时代了，但是如果大家能理解透彻我们遇到的问题，相信大家对 Zabbix 的内部实现会有更深入的了解。

22.1 Duplicated Host问题

这是 1.8.8 时代的一个问题，当 Zabbix 中出现两个 hostname 相同的 Host 或者名字相同的 Proxy 时，Zabbix 就会自动退出。大家是不是觉得这个设计非常的奇怪呢？

说到这个问题的起因，是因为另一个 bug——ZBX-4045，具体内容见 <https://support.zabbix.com/browse/ZBX-4045>。ZBX-4045 提出的 bug，是因为有重复名字的 Host 或者 Proxy 会导致 DCsync_hosts() 方法异常，从而导致 Zabbix 异常退出。出错的堆栈信息如下。

```
13235:20110812:180024.049 === Backtrace: ===
13235:20110812:180024.050 9: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server(print_fatal_info+0x333) [0x8090927]
13235:20110812:180024.050 8: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server() [0x808f534]
13235:20110812:180024.050 7: [0x3de40c]
13235:20110812:180024.050 6: /home/rudolfs/zabbix/branches/18/src/
```

```

zabbix_server/zabbix_server (DCsync_configuration+0x116) [0x8088208]
13235:20110812:180024.050 5: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server (main_dbconfig_loop+0x8b) [0x8057f57]
13235:20110812:180024.050 4: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server (MAIN_ZABBIX_ENTRY+0x54f) [0x805690b]
13235:20110812:180024.051 3: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server (daemon_start+0x428) [0x808fcd9]
13235:20110812:180024.051 2: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server (main+0x1fc) [0x80563ba]
13235:20110812:180024.051 1: /lib/i386-linux-gnu/libc.so.6 (__libc_start_
main+0xe7) [0x3f5e37]
13235:20110812:180024.051 0: /home/rudolfs/zabbix/branches/18/src/
zabbix_server/zabbix_server () [0x8051481]

```

这个 ZBX-4045 的解决方案，是在代码中增加了对于重复名字 Host 和 Proxy 的判断，一旦出现，则 Zabbix 会自动退出。

当时使用的版本是 1.8.8，因为我们不敢贸然升级，所以想在代码层把这个逻辑给解决。

代码的位置在 src/libs/zbxdbcache/dbconfig.c 中，具体如下。

```

if (NULL != host_ph)
{
    if (0 == found || sync_num == host_ph->sync_num)
    {
        /* duplicate hosts or proxies found */
        zabbix_log (LOG_LEVEL_CRIT, "Error: duplicate %s [%s] found.
        Exiting...",
            HOST_STATUS_MONITORED == host_ph->status ? "hosts" : "proxies",
            host_ph->host);
        exit (FAIL);
    }
    host_ph->host_ptr = host;
    host_ph->sync_num = sync_num;
}

```

我们把中间的一个判断全部注释掉，相当于把这个功能关闭了。

这个奇怪的功能只存在了一个小版本——正好是我们使用的 1.8.8，在 1.8.9 就移除了这个功能：

For version 1.8.9

The shutdown upon detection of duplicate hosts, introduced in 1.8.8, has been removed.

22.2 拼接大SQL问题

在 PPTV 时，Zabbix 后端的数据库是 Oracle。DBA 发现，Zabbix 会将 SQL 拼成一个 “begin...end” 的大 SQL 执行，影响性能，如果能将这些 SQL 分成一条一条 SQL 去执行，性能会好很多。

这是 Zabbix 本身在处理 SQL 时的一个方式。当执行的 SQL 长度超过 Zabbix 设定的阈值，并且数据库是 Oracle 时，Zabbix 会在一批 SQL 的开始加上 “begin”，末尾加上 “end”，将它们放到一个事务里让数据库处理，具体处理的代码逻辑如下。

```
void DBexecute_overflowed_sql(char **sql, int *sql_allocated, int *sql_offset)
{
    if (*sql_offset > ZBX_MAX_SQL_SIZE)
    {
#ifdef HAVE_MULTIROW_INSERT
        if (',' == (*sql)[*sql_offset - 1])
        {
            (*sql_offset)--;
            zbx_snprintf_alloc(sql, sql_allocated, sql_offset, 3, ";\n");
        }
#endif
#ifdef HAVE_ORACLE
        zbx_snprintf_alloc(sql, sql_allocated, sql_offset, 6, "end;\n");
#endif
        zabbix_log(LOG_LEVEL_WARNING, "OverFlow SQL: %s", *sql);
        DBexecute("%s", *sql);
    }
}
```

```

    *sql_offset = 0;
#ifdef HAVE_ORACLE
    zbx_snprintf_alloc(sql, sql_allocated, sql_offset, 7, "begin\n");
#endif
}
}

```

可以看到，这个机制是针对 Oracle 的，所以我们又中招了。

我们解决的方案是在执行 SQL 的入口处，判断 SQL 是否带有“begin”，如果有，就将它分割成一条一条小的 SQL 去执行。有兴趣的同学可以到 https://github.com/baniuyao/Zabbix_PPTV/commit/57dc865555e3224118e9b0daf6018c95266a0c95#diff-becfbeb5e63bc27ff1669364a34b1f6b 查看这一次 commit 的 diff，就知道具体的变动了。

22.3 nextid问题

nextid 是当时影响性能最大的一个问题。Zabbix 的各个资源在数据库里都有自己的 id，特别是 eventid。每一个 Event 的生成，都需要一个 eventid。而 Event 的生成又特别频繁，因为每一个 Item 有新的数据时，如果有 Trigger 和这个 Item 关联，都会生成一个 Event。那么 EPS (events per seconds) = VPS * N，其中 N 是一个 Item 平均和多少个 Trigger 关联的数字，可以这样说，EPS 是 VPS 的数倍。而 Zabbix 对于 nextid 的分配算法，问题非常大，我们先看下 Zabbix 是如何实现 nextid 算法的。

在数据库中有一张表叫 ids，它记录了每一个资源下一个 id 是什么，形如：

nodeid	table_name	field_name	nextid
0	acknowledges	acknowledgeid	2
0	actions	actionid	7
0	application_template	application_templateid	68
0	applications	applicationid	501
0	auditlog	auditid	431
0	auditlog_details	auditdetailid	140

	0	conditions	conditionid	10	
	0	dchecks	dcheckid	6	
	0	drules	druleid	5	
	0	expressions	expressionid	7	
	0	functions	functionid	13235	

在计算 Event 的 nextid 时, 步骤如下。

(1) 从 ids 表中取出 event 表的 nextid。

(2) 如果结果为空, 而且 events 表为空, 那么就将初始值插入 ids 表。如果由于并发造成的插入失败, 那么就会更新这条记录, 将 nextid 更新为 nextid+1。继续下一次循环。

(3) 如果结果不为空, ret1 等于 event 表的 nextid。这时 Zabbix 会更新 nextid=nextid+1, 然后再从 ids 表中取出最新的 event 表的 nextid 作为 ret2。在得到 ret1 和 ret2 后, Zabbix 会判断 ret2 是否等于 ret1+1, 如果相等, 说明这一次获取 nextid 是成功的, 如果不相等, 就再从来一遍。

这个逻辑在高并发的情形下, 在 Oracle 中会产生大量的 TX 锁, 非常影响性能。我们的解决方法是使用 Oracle 自带的 sequence 来获取这个 id。sequence 我们可以简单理解为一个自增序列。增加的方法非常简单, 代码如下。

```

zbx_uint64_t DBget_seq_maxid(const char *tablename)
{
    zbx_uint64_t ret;
    DB_RESULT result;
    DB_ROW row;
    result = DBselect("select eventid_sequence.nextval from dual");
    row = DBfetch(result);
    ZBX_STR2UINT64(ret, row[0]);
    //zabbix_log(LOG_LEVEL_INFORMATION, "***** GETTING SEQ MAX ID:%d
    *****", ret);
    DBfree_result(result);
    return ret;
}

```

22.4 在 Zabbix 中打印日志

前几节是如何解决问题的方法和思路，这一节，主要介绍如何在 Zabbix 中打印日志，这是非常重要的方法。C 语言不像 Python，它需要编译，在修改 Zabbix 代码时最头痛的就是调试了，每次都要编译安装好 Zabbix，然后看日志。所以尽可能详尽的日志能减少编译安装的次数。

Zabbix 打印日志非常简单，使用的方法是 `zabbix_log`，它的第一个参数是 `log` 的等级，第二个参数是日志的内容，如果日志内容中有占位符，那么后面接着的参数就是占位符的内容。比如 `zabbix_log (LOG_LEVEL_DEBUG, "debug")` 即表示为 `DEBUG` 等级的日志，打印出 `"debug"`。使用占位符是形如 `zabbix_log (LOG_LEVEL_DEBUG, "debug %d", 10)` 这种。

日志等级有下面几种：

- ◎ `LOG_LEVEL_CRIT`
- ◎ `LOG_LEVEL_ERR`
- ◎ `LOG_LEVEL_WARNING`
- ◎ `LOG_LEVEL_DEBUG`
- ◎ `LOG_LEVEL_INFORMATION`

第 23 章

PPTV的Zabbix监控体系

前面介绍了很多 Zabbix 的应用场景，“养兵千日用兵一时”，本章主要向大家介绍 PPTV 的监控体系为例，给大家一个完整的实战说明。前半部分主要介绍 PPTV 基于 Zabbix 进行的二次开发，后半部分介绍 Zabbix 在整个监控体系中的角色和作用。

23.1 Python Zabbix API

在 PPTV 的运维开发团队，主要的开发语言是 Python，因为 Python 入门简单，开发快速，而且是一个“电池都包括”的语言（表示 Python 的各种库应有尽有）。Zabbix 虽然用起来不算困难，但都需要鼠标操作，没办法集成到命令行中。Zabbix 本身提供了 API 接口，但是封装的不是很好，不是面向对象的建模，要记忆这么多方法也不容易。因此，笔者参照其他人开发的 Zabbix API，用 Python 写了一个模块——ZabbixPythonApi。它的作用是在 Zabbix API 上根据 Zabbix 对象封装了一层，让我们在使用 API 的时候是基于对象的。ZabbixPythonApi 已经开源在 Github 上了，地址为：<https://github.com/baniuyao/ZabbixPythonApi>。这个 API 是基于 Zabbix 1.8.8 开发的，在 Zabbix 2.2 也可以运行。

使用起来很简单，首先需要引入我们的 API 模块：

```
from zapi import ZabbixAPI
```

接着是登录 Zabbix：

```
zapi = ZabbixAPI(url='http://your.zabbix.address', user='admin',
```

```
password='zabbix')
```

```
zapi.login()
```

然后就是针对资源的操作，比如查找名字为“HostABC”的 host：

```
zapi.Host.find({ 'hostname': 'HostABC' })
```

返回值如下。

```
{'available': '1',
'disable_until': '0',
'error': u'',
'errors_from': '0',
'flags': '0',
'host': 'HostABC',
'hostid': '10108',
'ipmi_authtype': '-1',
'ipmi_available': '0',
'ipmi_disable_until': '0',
'ipmi_error': u'',
'ipmi_errors_from': '0',
'ipmi_password': u'',
'ipmi_privilege': '2',
'ipmi_username': u'',
'jmx_available': '0',
'jmx_disable_until': '0',
'jmx_error': u'',
'jmx_errors_from': '0',
'lastaccess': '0',
'maintenance_from': '0',
'maintenance_status': '0',
'maintenance_type': '0',
'maintenanceid': '0',
'maintenances': [],
'name': 'HostABC',
'proxy_hostid': '0',
'snmp_available': '0',
```



```
'snmp_disable_until': '0',
'snmp_error': u'',
'snmp_errors_from': '0',
'status': '1',
'templates': [{'hostid': '0', 'templateid': '0'}]}}
```

是不是返回的属性特别多？Zabbix API 会返回这个搜索到的 Host 的所有属性。可以在 find 方法中加入一个参数，使得 find 方法只会得到一个指定的属性，如：

```
zapi.Host.find({'name': 'HostABC'}, attr_name='hostid')
```

这样返回的就仅仅是搜索到的 Host 的“hostid”这个属性的值了：

```
['10108']
```

通过这个例子，大家应该能举一反三了吧。使用 API 时传递的参数（比如上面例子中的 name），可以参考 Zabbix 的官方文档。

23.2 Spider——服务器添加Zabbix监控

使用监控系统最大的问题，就是添加监控。在 PPTV 使用 Zabbix 的初期，添加监控非常麻烦。需要根据服务器上面运行的应用程序，去添加对应的监控。比如运行了 Redis 则要增加对 Redis 的监控，运行了 Nginx 需要添加对 Nginx 的监控。一台服务器还可以手动操作，如果 100 台、1000 台呢？而且，只要是人进行的操作，就是可能会出错的地方。

除了需要人力工作，容易出错外，手动添加服务器监控最大的问题就是无法和自动化运维的其他系统集成。理想的状态是一台服务器上线后，应该能自动添加监控；一台服务器下线，应该自动取消监控。很多公司应该使用的是类似“工单”的东西来进行流程上的控制，当服务器上线时，一张“添加监控工单”会发送给对应的运维工程师，然后运维工程师去进行操作。这一环，会变成整个自动化运维的瓶颈。

面对这种问题，Zabbix 推出了 auto discovery 功能，它设计的初衷就是解决上面提到的自动添加监控的问题。它的设计非常好，但有一些局限，比如它不支持从进程去判断服务器上存活了什么应用，只能从 Zabbix Server 去侦测 Zabbix Agent 上运行了哪些端口，从而去判断运行了哪些程序。而且，自动侦测使用 IP 范围去探测服务器，这个粒度太粗了，很有可能会造成误操作。比如服务器下线这个动作，如果使用了自动侦测，它会一直侦测这个网段，那么这里就隐含了

一个限制：删除监控会发生在服务器关机之后（我们使用 Action 来控制超过多久的服务器自动移除监控），这样就会产生误报警。假设监控了一台 Nginx 服务器，现在要下线了，那么确定流量切走以后，会关闭 Nginx 进程，然后关机。而 Zabbix 要在关机后才能删除监控，那么在关闭 Nginx 进程到关机的这段时间，Zabbix 会认为 Nginx 进程已经消失了，这是个问题，会报警。而实际上，这个根本不需要报警。

基于这些需求，笔者开发了 Spider，它的作用简单来说就是根据服务器的 hostname，对运行的应用加入对应的监控，而需要的只是一个 IP。

Spider 界面如图 23-1 所示。

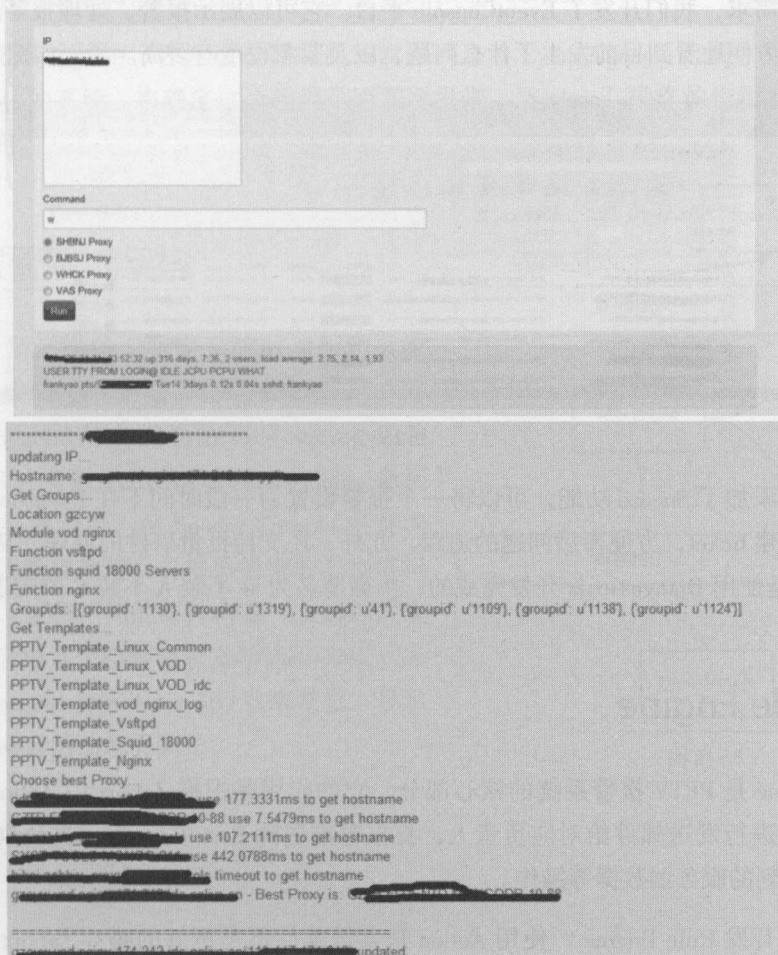


图23-1

23.3 Event Console

在查看报警时，Zabbix 做得不是很好。一是每一个报警的维度很少，只能看某个服务器的某个报警，顶多加上一个报警的等级。另外展示报警的方式也不好，不能让人很好地发现问题。而且，虽然 Zabbix 有 ACK 功能，但太过简陋，它只能知道某个人在什么时候 ACK 了这个请求。

在 PPTV 时，除了上海的兄弟们，武汉还有 24 小时值班的工程师，他们一般是第一个收到 Zabbix 报警的人，他们希望在接收到报警后，对于简单的问题可以直接远程执行命令，或者自动恢复，比如重启 PHP 进程之类。

基于这些需求，我们开发了 EventConsole 平台。它可以展示报警，处理报警。在上面运维工程师可以很方便地看到目前发生了什么问题，以及紧急程度有多高。界面如图 23-2 所示。

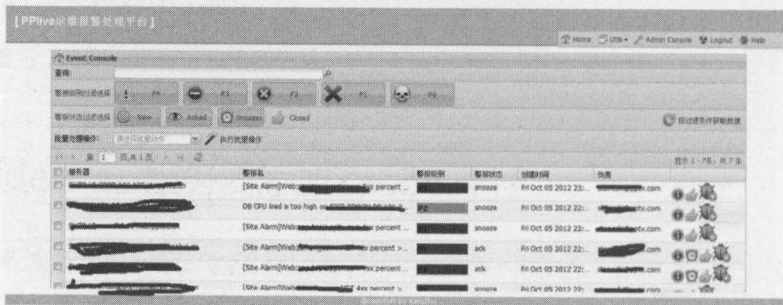


图23-2

我们同时添加了 Snooze 功能，可以将一个报警设置为一段时间不在 EventConsole 中显示，还可以直接创建 ticket，方便事后问题的追踪。此外，还支持批量事件的处理，功能非常强大。EventConsole 是使用 Django+extjs 开发完成的，由微博名为 @ 不装 A 不装 C 的同事负责。

23.4 Rule Engine

Rule Engine 是 PPTV 报警系统的核心部分。它的作用是根据 Zabbix 中 Action 的信息，组织报警数据，进行发送邮件给对应负责人，将报警推送到 EventConsole，自动执行一些命令，收集出问题时刻的服务器数据等操作。

为什么要开发 Rule Engine？使用 Action 来发送邮件不是挺好的吗？Zabbix 支持很多宏，可以在报警邮件中发送很多东西。我们先撇开 Zabbix，想象一下，如果你收到一封报警，你最

希望里面有些什么内容。以我个人来说,我想看到出问题这台服务器的一些基本系统数据,比如 CPU 负载、内存、哪些进程在跑等。我还想知道出问题的这个监控点的历史数据是怎样的。如果能根据报警内容的不同,事先做一些分析,那是更好的。

Rule Engine 就是为了满足这些需求而设计的。Zabbix 的 Action 不再发送邮件,而是将必要的信息,比如 itemid、triggerid 等传递到后端的 rule_engine_agent, rule_engine_agent 将这条消息发送到 ActiveMQ,然后 rule_engine 从 ActiveMQ 拿到消息,根据报警的类型,去组合不同的报警信息。比如报警内容为 Linux 服务器内存不够了,那么就需要内存使用的排行;如果报警是磁盘 I/O 高,那么就需要消耗 I/O 的排行。当组合好这个报警信息后,一是要发送邮件给对应的运维和研发,二是将消息发送到 Event Console。

在这个处理过程中,在发送报警邮件和发送消息给 Event Console 之前,会访问一个叫做 Snooze Console 的系统,来确定这个报警是否需要发送。“Snooze”操作的作用就是停止某一类报警一段时间。

23.5 报警系统架构

图 23-3 所示是整个报警系统的架构。报警的消息从触发 Action 开始,将需要的信息(比如 itemid, triggerid 等)发送给 Rule Agent, Rule Agent 将消息转发到 ActiveMQ。Rule Engine 会从 ActiveMQ 中抓取消息,从 Snooze Console 中查询这个报警是否需要处理,再根据报警的类型进行不同的处理,组合不同的邮件内容,发送邮件或者短信。然后把这个报警消息发给 ActiveMQ。之后,Event Console 会从 ActiveMQ 获取消息,显示在 Event Console 的前端界面中。

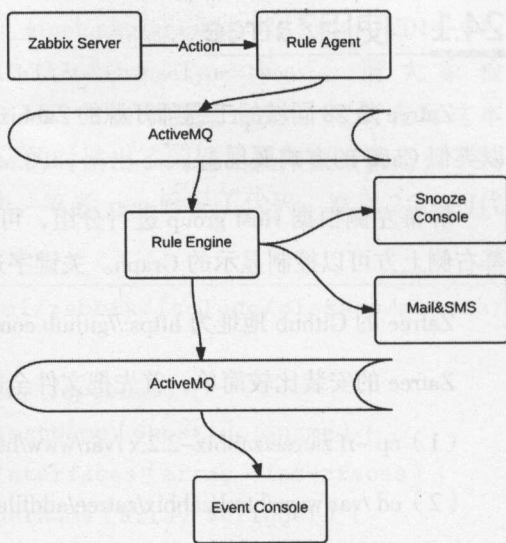


图 23-3

第 24 章

Zatree

24.1 使用Zatree

Zatree 是 58 同城的工程师开源的 Zabbix 前端插件,它的作用是将 Zabbix 的 Graph 组合起来,以类似 Cacti 的方式展现。

屏幕左侧根据 Host group 进行分组,可以选择其中的某台服务器显示它所有的 Graph。屏幕右侧上方可以控制显示的 Graph。关键字这里可以进行搜索,多个关键字可以使用逗号隔开。

Zatree 的 Github 地址为 <https://github.com/spide4k/zatree>。

Zatree 的安装比较简单,首先把文件全部复制到 Zabbix 前端目录下,包括:

- (1) `cp -rf zatree/zabbix-2.2.x /var/www/html/zabbix`
- (2) `cd /var/www/html/zabbix/zatree/addfile`
- (3) `cp -f CLineGraphDraw_Zabbix.php CGraphDraw_Zabbix.php CImageTextTable_Zabbix.php $ZABBIX_PATH/include/classes/graphdraw/`
- (4) `cp -f zabbix.php zabbix_chart.php $ZABBIX_PATH/ cp -f CItemValue.php $ZABBIX_PATH/api/classes/`
- (5) `cp -f menu.inc.php $ZABBIX_PATH/include/`

(6) `cp -f main.js $ZABBIX_PATH/js/cp -f API.php $ZABBIX_PATH/include/classes/api/`

如果大家懒得敲代码，可以到笔者 Github 上的 gist 中找到安装脚本：<https://gist.github.com/baniuyao/10016098>，其实就是把这些命令放到一起了。

复制好以后，需要将 `/var/www/html/zabbix/zatree/zabbix_config.php` 中的 Zabbix 前端用户名和密码写进去，因为 Zatree 的一部分操作要和 Zabbix API 交互，获取需要的信息，因此要先登录，如下。

```
vi /var/www/html/zabbix/zatree/zabbix_config.php
'user'=>'xxx',           //Web 登录的用户名
'passowrd'=>'xxx',       //Web 登录的密码
```

笔者在安装中碰到了下面这个问题：前端可以进入 Zatree，但从 console 来看，报 500 错误，查看了 PHP 日志后，发现错误为“Call to undefined method CMacrosResolverHelper::resolveItemNames() in `/var/www/html/zabbix/include/classes/api/CLineGraphDraw_Zabbix.php` on line 107, referer: `http://192.168.201.234/zabbix/zatree/big_graph.php?graphid=518&stime=2014-04-07+13%3A31%3A36&endtime=2014-04-07+14%3A31%3A36&changeType=1hour`”。请大家检查 Zabbix 的前端版本，因为这里需要 `resolveItemNames` 方法来获取 itemid。笔者在写这本书的时候，前端界面的版本为 2.2.1，在安装 Zatree 的时候出了问题，具体报错就是找不到 `CMacrosResolverHelper` 模块的 `resolveItemNames` 方法。笔者 `grep` 后看了代码，发现 2.2.1 的代码中确实没有这个方法：

```
grep 'function resolve' /var/www/html/zabbix/include/classes/macros/
CMacrosResolverHelper.php

public static function resolve(array $options) {
public static function resolveHttpTestName($hostId, $name) {
public static function resolveHostInterfaces(array $interfaces) {
public static function resolveTriggerName(array $trigger) {
public static function resolveTriggerNames(array $triggers) {
public static function resolveTriggerDescription(array $trigger) {
public static function resolveTriggerDescriptions(array $triggers) {
public static function resolveTriggerNameById($triggerId) {
public static function resolveTriggerNameByIds(array $triggerIds) {
public static function resolveTriggerReference($expression, $text) {
public static function resolveEventDescription(array $event) {
```

```
public static function resolveGraphName($name, $items) {
public static function resolveGraphNameByIds($data) {
```

这时笔者去下载了 Zabbix 2.2.2 版本，发现这个方法又有了，其中缘由无从而知，估计是 Zabbix 团队在重构前端代码时的一次修正，而 58 同城的工程师没有发现这个小版本的变动。我们看 Zabbix 2.2.2 版本中的代码：

```
grep 'function resolve' ~/frankyao/zabbix-2.2.2/frontends/php/include/
classes/macros/CMacrosResolverHelper.php

public static function resolve(array $options) {
public static function resolveHttpTestName($hostId, $name) {
public static function resolveHostInterfaces(array $interfaces) {
public static function resolveTriggerName(array $trigger) {
public static function resolveTriggerNames(array $triggers) {
public static function resolveTriggerDescription(array $trigger) {
public static function resolveTriggerDescriptions(array $triggers) {
public static function resolveTriggerNameById($triggerId) {
public static function resolveTriggerNameByIds(array $triggerIds) {
public static function resolveTriggerReference($expression, $text) {
public static function resolveTriggerExpressionUserMacro(array
$trigger) {
public static function resolveEventDescription(array $event) {
public static function resolveGraphName($name, array $items) {
public static function resolveGraphNameByIds(array $data) {
public static function resolveItemNames(array $items) {
public static function resolveItemKeys(array $items) {
public static function resolveFunctionParameters(array $data) {
```

在将前端升级为 2.2.2 后问题解决。

24.2 Zabbix二次开发和重新开发监控系统的选择

我是从 2011 年开始使用 Zabbix 的，现在已经是 2016 年（本节为 2016 年新增内容），一个开源产品有这么长的生命周期，到如今依然非常流行，说明监控系统是运维不可或缺的。但从

另一方面来说,可能 Zabbix 已经到了需要大刀阔斧修改的时候了,Zabbix 大版本来到 3.0 也说明了这个问题。我们看看大数据一些产品的发展,Storm、Spark、Presto 等各种新技术层出不穷,监控系统是不是也应该有一些革新呢?

最近几年,国内使用 Zabbix 的大公司,一般都会走两条路,一个是改造 Zabbix,一个是重新开发。改造 Zabbix 主要集中在改造它的存储层。Zabbix 的数据是存储在传统的 RDBMS 中的,而传统的 RDBMS 并不非常适合运维的海量数据的场景。前两年我了解到,美团的做法是将 Zabbix MySQL 中的数据再写入 OpenTSDB,然后重新开发了 Zabbix 的前端,数据全部从 OpenTSDB 来获取,相当于只是将 Zabbix 当作数据收集和报警配置的工具。2015 年我在唯品会的最后一段时间里,尝试将 HBase 替换掉原生的 RDBMS,花了非常大的努力,完成了这个工作,但是对 Zabbix 的代码有极大的侵入,几乎将数据存储层的代码全部重写了。这非常不利于之后 Zabbix 的升级和维护。小米是重新开发了 open-falcon 监控系统,他们使用 rrdtool 来作为数据存储引擎,个人感觉,rrdtool 也并不是一个大数据场景下很好的解决方案。

和一些朋友沟通中也发现,其实很多公司都到了这个岔路口。当时开始使用 Zabbix 的时候,是从没有监控系统到有监控系统,这个从 0 到 1 的变化会带来非常多的好处,但用了一段时间后,发现 Zabbix 只是做到了 60 分或者 70 分,想要把监控系统做到 90 分,光靠 Zabbix 本身是非常困难的。像我,最早在 PPTV 是通过围绕在 Zabbix 周边的各种工具来完成这个需求的。但过去这么多年,这个真的是好的选择吗?

我们首先来看看一个广义的监控系统,它由这几部分组成:

- 数据收集
- 数据存储
- 数据报警
- 数据展现

无论是怎样的监控系统,都脱离不开这 4 个部分,其中难点在于数据存储。比如 Zabbix 就因为使用了 RDBMS 限制了它的扩展能力。

本节的标题是选择二次开发 Zabbix 还是开发新的监控系统,我的看法是这样的,如果对于前面提到的“数据收集”、“数据报警”、“数据展现”的功能不满意,那么我们可以开发一些工具来增强 Zabbix 的功能。但如果是 Zabbix 的性能不能满足大规模的需求,那有两种可能:

- 公司的服务器规模非常大。
- 如果服务器规模不大,那就是因为监控项设置有问题。

第2种情况是可以通过优化监控项来完成的，这里不做赘述。针对第1点，如果公司的规模真的已经增长到 Zabbix 无法掌控的地步，那就说明公司的发展非常好，同样，需要更强大的技术来支持，这个时候，可以考虑自己来开发监控系统了。因为研究 Zabbix、改造 Zabbix 和维护 Zabbix，已经会耗费非常多的精力了。

那么我们如何入手开发新的监控系统呢？我的建议是不要想开发一个系统能把 Zabbix 替换掉，我们应该先逐步替换 Zabbix 的各个组件。最容易做的是“数据收集”和“数据展现”这两部分，利用 Zabbix trapper 能方便地用自己的工具来向 Zabbix 发送数据。对于“数据展现”，Zabbix 的数据都存储在 RDBMS 中，可以方便地读取。

更深入一些，除了上面说到的4个组件，监控系统还可以有“数据分析”这一步，它的作用是分析数据，找到问题。这个需要和大数据结合，在唯品会做 HBase 和 Zabbix 集成的工作时，也研究了 etsy 公司的一些相关数据查找的工作，做到可以从 Zabbix 中根据一个数据图形找到图形趋势类似的数据，这个对于 Debug 问题是非常有用的，但因为 Zabbix 有海量的数据，这样的分析，我们是一定要借助大数据的工具的。

第 25 章

Zabbix第三方插件

这一章，向大家介绍两个 Chrome 使用的 Zabbix 插件。

25.1 Chromix

Chromix 的功能是可以直接在 Chrome 中查看 Zabbix 的报警情况，它的地址在 <https://chrome.google.com/webstore/detail/chromix/odjpdjeegacmncmodjbeboldofhljjif>，使用起来非常简单，安装完后单击 Chrome 中的按钮后即可配置相应的 Zabbix，如图 25-1 所示。

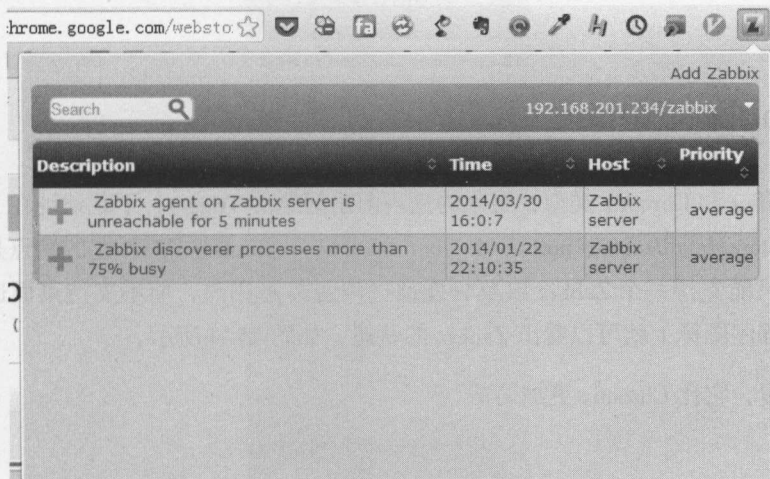


图 25-1

同时它还支持调用 Chrome 的桌面通知进行报警，并且可以配置多套 Zabbix。如图 25-2，笔者配置了两套 Zabbix。

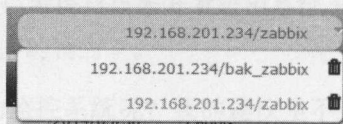




图25-2

配置完成后的结果如图 25-3 所示。

Add Zabbix

Search 

192.168.201.234/zabbix 



Description	Time	Host	Priority
 Zabbix agent on Zabbix server is unreachable for 5 minutes	2014/03/30 16:0:7	Zabbix server	average
Comments :			
Error :			
 Zabbix discoverer processes more than 75% busy	2014/01/22 22:10:35	Zabbix server	average

图25-3

25.2 Zabbix Notifier

Zabbix Notifier 和 Chromix 类似，展示的是当前出问题的 Trigger，下载地址在 <https://chrome.google.com/webstore/detail/zabbix-notifier/ikeijbmpddnkaeejokgifiocbcijjfo>，它的缺点是不支持多个 Zabbix 实例，只能支持一个 Zabbix 以及只能通过声音提示用户，而不支持桌面通知。它的优势在于在扩展程序图标上就可以看出 Zabbix 的状态，如图 25-4 所示。

从界面来看，它比 Chromix 更加简洁。

gigatec Zabbix Notifier: Overview

System	Description	Priority	Age
server20	ssh server alive on server20	High	36d. 1h. 51m
server11	1 Updates (Debian Security)	Average	3d. 19h. 38m
server08	1 Updates (Debian Security)	Average	3d. 19h. 38m
server19	Postfix: Too many held mails on server19	Warning	60d. 8h. 43m
server07	MySQL: High Number of MySQL queries per second on server07	Warning	111d. 0h. 53m
server11	18 Updates (Debian System)	Information	9d. 15h. 38m
server08	21 Updates (Debian System)	Information	9d. 15h. 38m
server18	2 Updates (Debian System)	Information	49d. 9h. 16m

Overview | Settings | Open Zabbix

图 25-4

25.3 手机端 Zabbix App

做过运维工程师的都知道，我们的手机基本是 24 小时待命的，一旦有什么问题，需要马上处理。Zabbix 自己是没有手机客户端的，由于有丰富的 API，有很多第三方开发的 App 可以使用。由于条件有限，笔者只测试了 Zabbix 网站上第三方插件中介绍的两款 iOS 的 App——一个是 ZBX Mobile，另一个是 zabbixkit。

25.3.1 ZBX Mobile

ZBX Mobile 的界面如图 25-5 所示，这是展示目前有哪些 Trigger 报错的。

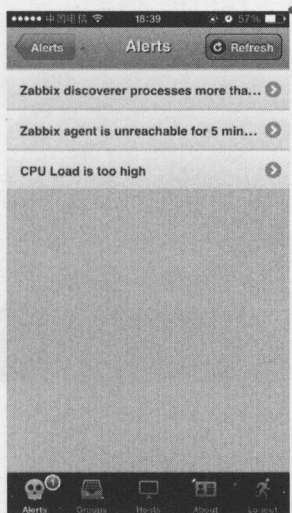


图 25-5

显示 Host 的 Graph 列表，如图 25-6 所示。



图25-6

具体的 Graph，如图 25-7 所示。

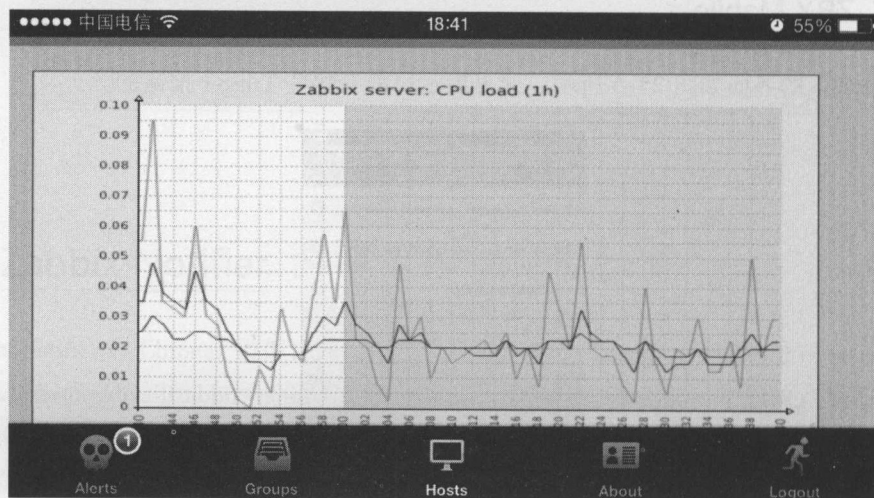


图25-7

但是 ZBX Mobile 不是很稳定，有时候会报错，如图 25-8 所示。



图 25-8

25.3.2 Zabbkit

Zabbkit 是笔者个人比较喜欢的 App，上一节提到的 ZBX Mobile 界面还没有过渡到 iOS7，比较丑陋，而且会 crash。Zabbkit 则没有这些问题。

首先是 Trigger 的总览，它有不同颜色来区分不同严重程度的报警，而且还在 Trigger 名字下方标注了 Trigger 来自哪个 Host，如图 25-9 所示。

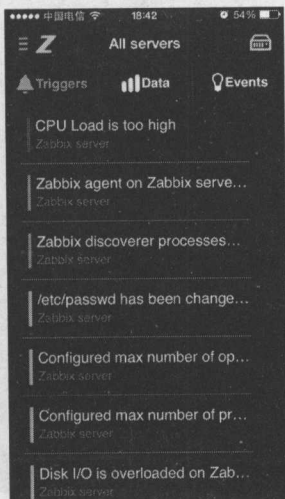


图 25-9

Events 列表, 如图 25-10 所示。

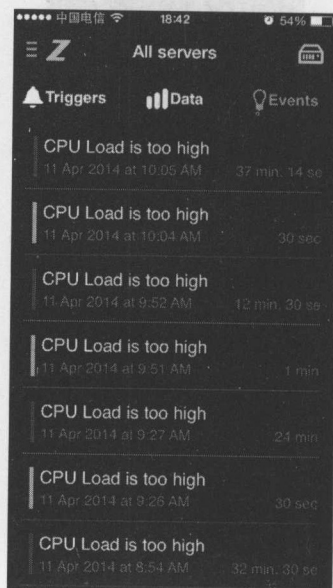


图25-10

Host group 列表, 单击进入后可以选择服务器查看监控项的值, 如图 25-11 所示。



图25-11

单击右侧的图表状的按钮，可以看到 Graph，如图 25-12、25-13 所示。

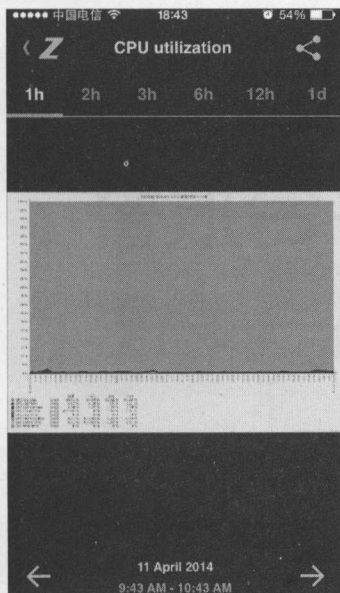


图25-12

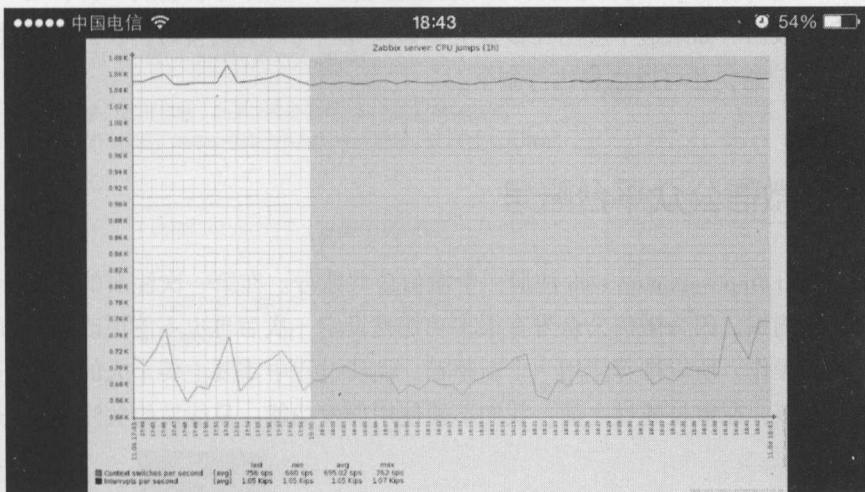


图25-13

笔者使用 Zabbix 时感觉很不错，唯一的问题就是要连接上 Zabbix 必须手机能连接公司内网，但在外面不可能时刻连接着公司内网。

第 26 章

微信公众平台报警

本章会介绍如何使用微信的公众平台进行报警。虽然现在智能手机普及了，但是邮件还是不能做到实时推送。短信如果不使用邮件转发手机短信这种功能的话，也比较难实现。现在人人有微信，还能做到实时性的推送。最重要的一点是，邮件报警和短信报警都是单方面的，不是可交互的。微信的公众平台有一个交互过程。比如输入“周围的饭店”，某个公众平台会告诉我“1. ××× 饭店，2. ××× 饭店，3. ××× 饭店”，然后可以输入“1”、“2”或者“3”来获取更详尽的数据。如果这个应用在报警上会如何呢？当收到报警时，我们能通过微信，获取 Zabbix 的详细信息，还可以远程执行命令。

26.1 申请微信公众平台账号

首先进入 <http://mp.weixin.qq.com> 注册一个微信公共账号。在第一次注册的时候，最好申请一个新的邮箱来测试，因为微信公众号有很多功能是设定一次后再也不能更改的，防止大家在不熟悉的时候误操作，所以推荐使用一个新邮箱。注册过程中需要上传个人的身份证照片，需要 7 个工作日审核，在审核通过之前，我们不能使用群发功能，只能发给关注这个公众账号的某个用户。

通过审核以后，在微信公众平台账号后面，可以看到多了几个栏目，分别是“服务”，“统计”还有“高级功能”。

26.2 配置微信公众平台账号

我们使用微信来做报警，主要是想调用一个 API 接口，把报警的内容发送给它，然后通过公众平台，把报警发给对应的人。最重要的就是这个 API 接口。可以从如图 26-1 所示的地方进入。

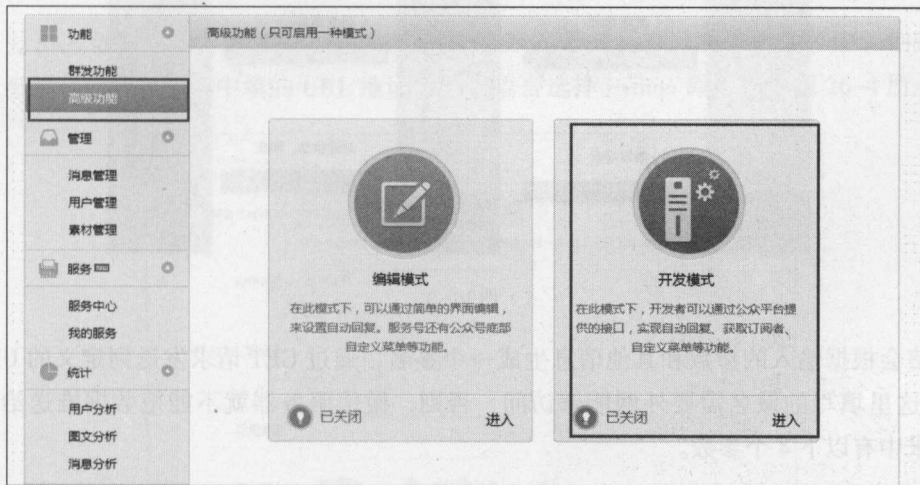


图 26-1

然后申请成为开发者，如图 26-2 所示。

请填写接口配置信息，此信息需要你拥有自己的服务器资源。
填写的 URL 需要正确响应微信发送的 Token 验证，请阅读接入指南。

URL

Token

● 必选字段
什么是Token？

图 26-2

下面一步非常关键，URL 中要填写一个外网域名，而 Token 则是我们自定义的一个字符串。为什么需要外网域名呢？微信公众平台的工作方式是：当我们使用手机发送消息给公众平台的

时候，微信服务器接收到了我们的消息，根据公众平台的账号，将用户发送的消息转发给公众平台配置的域名，然后根据域名的返回结果，将信息反馈到用户的手机上。过程如图 26-3 所示（图来自网络）。

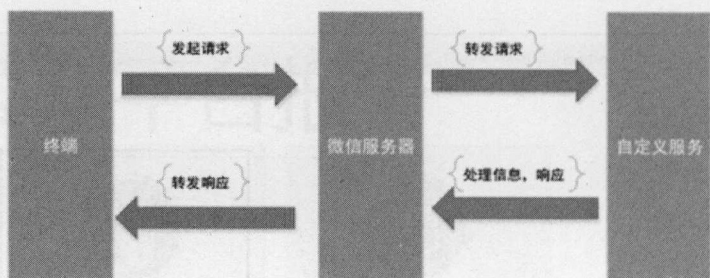


图26-3

微信会根据输入的参数和其他信息生成一个签名，通过 GET 请求发送到定义的 URL（所以 URL 这里填写的域名需要外网能够访问，否则，微信服务器就不能把数据推送给我们）。GET 请求中有以下 4 个参数。

- ◎ signature：加密签名。
- ◎ timestamp：时间戳。
- ◎ nonce：随机数。
- ◎ echostr：随机字符串。

当服务接收到 GET 请求后，需要根据内容，确认信息的正确性，确认的方法为：

- （1）将 token、timestamp，nonce 三个参数进行字典序排序。
- （2）将三个参数字符串拼接成一个字符串进行 sha1 加密。
- （3）开发者获得加密后的字符串与 GET 请求中的 signature 对比，如果相同则说明确认信息正确。

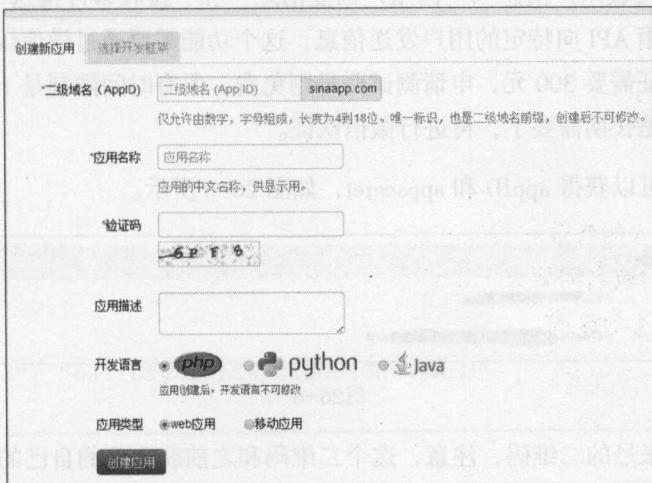
在确认正确后，就直接返回 echostr 的内容给微信服务器，这样就成为开发者了。

我推荐大家在测试的时候，使用 SAE（Sina App Engine，新浪推出的云服务，可以在上面开发代码，并且有 *.sinaapp.com 的二级域名，目前支持 Java、PHP 和 Python）来进行测试，一是因为公司的运维肯定不愿意搞一个外网服务器并且还给你个域名让你去测试微信公众平台。二是使用 SAE 非常方便，编写代码等全部可以在浏览器中完成。

26.2.1 使用SAE进行测试开发

笔者在进行微信公众平台的测试开发的时候，开始时找不到一个地方可以提供外网可以解析的域名，最后依靠 SAE 搞定了外网问题。下面一起看下如何使用 SAE 搭建一个简单的 Python 服务，使用 bottle.py，用来解析访问它的请求。

先去 SAE 创建应用，网址为 <http://sae.sina.com.cn>，登录后在“我的首页”中，创建一个应用，二级域名就是在微信中填的 URL 地址，开发语言选择 Python 即可，如图 26-4 所示。



创建新应用 选择开发框架

二级域名 (AppID) 二级域名 (AppID) sinaapp.com
仅允许由数字、字母组成，长度为4到18位，唯一标识，也是二级域名前缀，创建后不可修改。

应用名称 应用名称
应用的中文名称，供显示用。

验证码 验证码

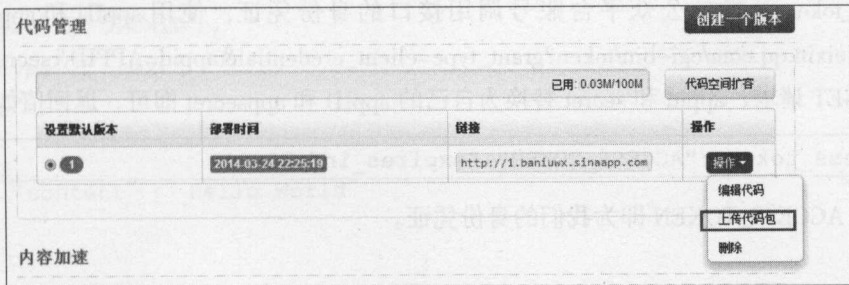
应用描述 应用描述

开发语言 ☒ php ☒ python ☐ java
应用创建后，开发语言不可修改

应用类型 ☒ web应用 ☐ 移动应用

图 26-4

创建完以后就可以部署代码了，笔者已经将一个可用的版本放在了 Github 上，有需要的同学可以自取，地址为：<https://github.com/baniu Yao/SaeWeiXin>。打包为 zip 后可以直接上传到 SAE 上使用，如图 26-5 所示。



代码管理

已用 0.03M/100M

设置默认版本	部署时间	链接	操作
<input checked="" type="radio"/> 1	2014-03-24 22:25:19	http://1.baniuwx.sinaapp.com	<div>操作 编辑代码 上传代码包 删除</div>

内容加速

图 26-5

最后在微信公众平台的配置中，URL 输入在 SAE 配置的二级域名；TOKEN 写 zabbix，可以自定义。需要注意的是，这里修改了，在 SAE 的 index.py 中，TOKEN 这个变量也需要统一。输入完毕后，就可以验证了。

26.2.2 申请测试账号

验证通过后，要申请测试开发账号，因为申请的只是普通的公众平台的账号，提供的 API 非常有限，只有“接收用户消息”、“向用户回复消息”和“接收事件推送”这三个基础接口。而我们的需求是调用 API 向特定的用户发送信息，这个功能需要通过微信认证，或者申请测试账号获得。微信认证需要 300 元，申请测试账号则免费，但有时间限制是 1 年，建议各位先申请测试账号，真的是长期需要了，再进行微信认证。

输入信息后，可以获得 appID 和 appsecret，如图 26-6 所示。

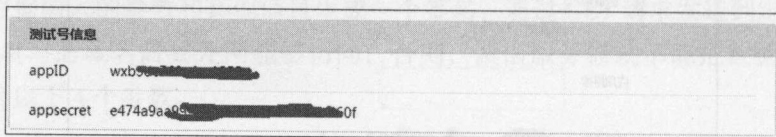


图26-6

同时获得测试账号的二维码，注意，这个二维码和之前获取到的自己的公众账号的二维码是不相同的，一定要再一次关注这里显示的二维码，用微信扫一扫后，关注“微信公众平台测试号”。

26.2.3 获取access_token

access_token 是微信公众平台账号调用接口的身份凭证，使用 appID 和 appsecret，向 https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET 发送 GET 请求，appid 和 secret 替换为自己的 appID 和 appsecret 即可，返回值如下。

```
{"access_token": "ACCESS_TOKEN", "expires_in": 7200}
```

其中，ACCESS_TOKEN 即为我们的身份凭证。

26.2.4 获取用户的openid

openid 是根据用户的微信号加密而成的，只能获取关注我们测试账号的用户的 openid。获取所有用户的 openid 的接口如下：

```
https://api.weixin.qq.com/cgi-bin/user/get?access_token=ACCESS_TOKENQ&next_openid=
```

其中，next_openid 表示一个偏移量，意思是从第几个关注用户开始查询，要查询全部的，就留空。返回值如下：

```
{
  "total": 1,
  "count": 1,
  "data": {
    "openid": [
      "OPENID"
    ]
  },
  "next_openid": "o37IDtySzTLZ6ZClUGosGWevrABM"
}
```

26.2.5 发送第一条文字消息

发送消息使用 POST 请求，目标 URL 是 https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=ACCESS_TOKEN，推送的是文字信息，POST 的内容如下。

```
{
  "touser": "OPENID",
  "msgtype": "text",
  "text": {
    "content": "Hello World"
  }
}
```

不出意外的话，我们的手机就能收到这个消息了。

26.3 微信接口请求次数限制

微信公众平台对于一些操作都做了每日请求次数的限制，在发送报警的过程中，用到了下面几个接口：

- (1) 请求 `access_token`。
- (2) 获取关注用户列表（用于获取 `openid`）。
- (3) 发送客服消息。

微信对于请求 `access_token` 没有限制，获取关注用户列表每日限额为 100 次，发送客服消息每日限额为 50000 次。

由于获取关注用户这一限制，我们不能频繁地获取到最新的用户列表，笔者的建议是，将需要发送报警人的信息由 Zabbix 维护，然后在数据库中存储一张“Zabbix 用户—微信 `openid`”的对应表，每 15 分钟通过获取关注用户接口拉取最新的数据填充。这样每一次在调用发送客服消息接口时，用户的 `openid` 直接从数据库中获取，避免频繁请求获取关注用户的列表。每 15 分钟请求一次，可以做到每天 $24 \times 60 / 15 = 96$ 次请求，低于 100 次。而且，晚上一般不会有人更新这个用户，我们可以做到工作时间 5 分钟刷新一次数据库内关系，非工作时间 30 分钟更新一次，保证了在工作时间中更加快速地生效。

微信公众平台本身是支持图片报警的，但笔者认为不应该让手机端的报警做的这么厚重，正确的流程是：工程师通过微信收到微信报警，如果自己有时间就应该马上处理，如果没有时间则通知其他工程师。

微信报警的目的是快速，使用图片一方面需要上传图片素材，另一方面增加了手机端的流量（这个是针对 2G 用户的），综合考虑，使用单纯的文字消息是比较合适的。

第 27 章

社区论坛

Zabbix 的社区目前不是非常活跃，大家交流主要是在 qq 群，比较国内活跃的群有“zabbix 交流群”92242469，还有个较为官方的“ZABBIX 中文论坛”62239409，Zabbix 的官方论坛 <https://www.zabbix.com/forum/> 也是比较活跃的地方，其中有中文版 <https://www.zabbix.com/forum/forumdisplay.php?f=26>，但相对 Zabbix 论坛，中文版比较冷清。笔者个人比较喜欢 Github 上的 Issue 方式来提问，因为它比较像 TODO list 的方式，一个问题结束后就可以 close 掉。后面有人如果要提问相同的问题，他可以先进行搜索，看是否已经有解答的方案。笔者在 Github 上建立了 ZabbixQuestionList 项目，就是这个目的，大家可以到 <https://github.com/baniuyao/ZabbixQuestionList/> 提问，笔者尽量做到每个问题都有解答。

对于提问，在此想多说几句。在群里面，有很多朋友对于 Zabbix 有很多问题，同时笔者也会经常收到朋友们的邮件。但是有些问题是可探讨性有些欠缺的，主要有以下几种。

- (1) 问题空泛，比如 Zabbix 怎么安装，数据库怎么配。
- (2) 事先没有仔细阅读资料，问题比较老旧。
- (3) 没有深度思考，比如明明日志里告诉你数据库没有连接上，仍然询问为什么我的 Zabbix 启动不了。

笔者希望大家碰到问题，能够先自己思考，想一下解决办法，在思考了一段时间后，还是没有解决，再把问题提出让大家帮忙。有朋友问，我一开始就提问然后得到解决，那么不就需要浪费时间了吗？是的，这样做，是可以即时解决问题，但却错过了一个思考的机会。一个

好的问题，提问的人在思考的过程中已经获益颇多，而回答的人在和提问人一起讨论问题的时候，双方都是一个提高的过程。但如果一点都不思考，有问题就抛出来，很有可能是个非常简单的问题，这样问题解决后，下次碰到类似的情况，还是不会思考，只会提问。久而久之，就形成了思维定势：有问题一提问，而不是有问题一思考一提问，最后在自己的技术领域中一点都没有提高。

附录

Zabbix自带宏

Zabbix 自带的宏非常多，而且也不是在任何地方都能使用，Zabbix 官网有一张表格，里面记录的是 Zabbix 现在支持的宏和使用的范围，地址在 https://www.zabbix.com/documentation/2.2/manual/appendix/macros/supported_by_location。在本附录中，笔者不会完整的将表格翻译一遍，这里主要会说明一些笔者认为常用的宏(比如 Inventory 就没有介绍的必要)。下面介绍的这些宏，大多数是在通知中可用，即报警信息中，至于具体的使用范围，请到上述 URL 中查询。

宏 名 称	说 明
{EVENT.ACK.HISTORY}	Event 的 ACK 的历史
{EVENT.ACK.STATUE}	Event 的 ACK 状态
{EVENT.AGE}	Event 产生到现在的时间，对于报警升级时的信息提示，十分有用
{EVENT.DATE}	Event 产生的日期
{EVENT.ID}	Event 的 id，和数据库中 events 表的 eventid 对应
{EVENT.RECOVERY.DATE}	Trigger 恢复时候产生的 Recovery 类型的 Event 产生的时间
{EVENT.RECOVERY.STATUS}	Recovery 类型 Event 的状态，返回的是字符串
{EVENT.RECOVERY.TIME}	Recovery 类型 Event 产生的时间
{EVENT.RECOVERY.VALUE}	Recovery 类型 Event 的状态，返回的是数字
{EVENT.STATUS}	Event 的状态，返回的是字符串
{EVENT.TIME}	Event 产生的时间
{EVENT.VALUE}	Event 的状态，返回的是数字
{HOST.IP<1-9>}	Trigger 中的 Expression 中的第 N 个 Host 的 IP。最多支持 9 个，可以这样使用 {HOST.IP1}
{ITEM.DESCRPTION<1-9>}	Trigger 中的 Expression 中的第 N 个 Item（下简称 Item）的 Description，即 Item 配置界面中配置的 Description
{ITEM.ID<1-9>}	Item 的 id，和 items 表中的 itemid 表一致
{ITEM.KEY<1-9>}	Item 的 key

续 表

宏 名 称	说 明
{ITEM.LASTVALUE<1-9>}	Item 最近一次获取的数据
{ITEM.NAME<1-9>}	Item 的名称
{ITEM.STATE<1-9>}	Item 的状态, 返回值是 “Not supported” 或者是 “Normal”
{NODE.ID<1-9>}	Trigger 中的 Expression 中的第 N 个 Item 所在 Host 属于的 Node (下简称 Node) 的 id
{NODE.NAME<1-9>}	Node 的名称
{PROXY.NAME<1-9>}	类似 {NODE.NAME<1-9>}
{TRIGGER.DESRIPTION}	Trigger 的 Description
{TRIGGER.EVENTS.PROBLEM.ACK}	Trigger 中被 ACK 的 Event 数量
{TRIGGER.EVENTS.PROBLEM.UNACK}	Trigger 中没有被 ACK 的 Event 数量
{TRIGGER.PROBLEM.EVENTS.PROBLEM.ACK}	和 {TRIGGER.EVENT.PROBLEM.ACK} 类似, 只是这里会多加一个限制, 返回的是所有 Problem 状态的 Trigger 的被 ACK 的 Event
{TRIGGER.PROBLEM.EVENTS.PROBLEM.UNACK}	同上, 所有 Problem 状态的 Trigger 中没有被 ACK 的 Event
{TRIGGER.EXPRESSION}	Trigger 的 Expression, 这个能够让我们知道报警的具体逻辑
{TRIGGER.ID}	Trigger 的 id, 和 triggers 表中的 triggerid 对应
{TRIGGER.NAME}	Trigger 的名称
{TRIGGER.NSERVERITY}	Trigger 的 Serverity 的数字形式。默认情况下, “0” 表示 “Not classified”, “1” 表示 “Information”, “2” 表示 “Warning”, “3” 表示 “Average”, “4” 表示 “High”, “5” 表示 “Disaster”。严重程度依次上升
{TRIGGER.SERVERITY}	Trigger 的 Serverity 的字符串形式
{TRIGGER.STATE}	Trigger 最近一次状态, 返回值是 “Unknown” 或者 “Normal”
{TRIGGER.STATUS}	Trigger 的状态, 返回的是字符串 “Problem” 或者 “OK”
{TRIGGER.TEMPLATE.NAME}	Trigger 所在的 Template, 如果返回的是 “UNKNOWN” 表示 Trigger 是定义在 Host 上
{TRIGGER.URL}	Trigger 的 URL, 即在 Trigger 定义时输入的 URL。比如对于某一个 Trigger 已经有了解决方法, 并且记录到了 wiki, 那么这里就可以记录下来。这样的话, 以后收到同样的报警以后就可以知道报警的解决方案是什么了
{TRIGGER.VALUE}	Trigger 的状态, 返回的是数字, “0” 表示 “OK”, “1” 表示 “Problem”

后记

终于到了后记部分了，感觉很有成就感，因为能够写到后记，就说明这本书已经完成了。在写这本书的过程中，我经常 would 想后记要怎么写。因为自古至今，做了什么大事，都喜欢要记录一番。就像古人打了胜仗要立碑述功一样。虽然没有那么隆重，但写这本书对我个人而言，实在具有人生里程碑的意义。

写书最大的感受是——累。因为自己限定了时间，我需要将大任务分割成小任务，然后再细化到每天要写的篇幅。一旦哪天有事或者状态不佳或者是才思枯竭，那这天应该完成的部分就要变成后面几天的压力了。

这个很像以前念书时候的暑假作业，一开始上来非常拼命，一天做两天的暑假作业，妄图一个月完成，然后第二个月就可以肆无忌惮地玩了。但根据我自己和同学的经验来看，大家基本都是一个套路——先拼命两个礼拜，完成很多作业后，就开始玩了。直到临近开学，才开始恶补。幸好这次写书不像如此，虽说一周中某一天会有延迟，但每一周的进度都是总体跟得上的。

我在 PPTV 使用的 Zabbix 版本是 1.8，写书时 Zabbix 的版本是 2.2。中间的时间我在唯品会做数据科学的基础架构的工作，和 Zabbix 没多大关系。为了写好这本书，查阅了很多文档，也和以前的同事交流了很多。结合诸多方面的帮助，才完成了这本书。离开一个技术一年，就有很多新东西要学习。不禁感叹，技术的发展真是非常迅速。

吃技术饭就意味着要“活到老，学到老”。在这我希望我能和各位一起保持对技术的热情，追逐技术的梦想。

最后，再一次感谢我的父母和亲爱的老婆，你们对我的支持是对我莫大的鼓励。谢谢你们。

姚仁捷

2014 年 6 月 16 日 于上海

程序员职业生涯的一些感悟

我们在做“产品”

我是个程序员，我相信买这本书的大多数也是程序员。在我工作的过程中，以及跟其他朋友聊天下来的感觉，我发现，很多程序员对于“产品”这两个字是没有概念的。这里提到的“产品”，不是一般意义上的“产品”——某个 App 是个产品，某个网站是个产品，某个活动是个产品。而是指的是程序员做的每一个“东西”。

我一直在做系统开发相关的工作，做出来的东西，大多数面对的用户是程序员。比如最早在 PPTV 做的监控系统，目前在唯品会做的日志平台。你做出来的每一个被他人使用的工具，都是一个“产品”，而且这个“产品”打上了你的 Logo。“产品”靠谱，说明你这个人靠谱，反之亦然。用户会把“产品”和你这个人绑定起来，因“产品”来对你进行加减法。

先可用，再好用

这 6 个字，是我对于产品的基本观点。大多数程序员都是完美主义者，巴不得开发的东西一个 Bug 都没有，然后界面酷拽炫，高大上，才能推出去给别人用。比如做个数据可视化工具，一定要把前端界面搞的跟科幻片一样才行，否则就是“不可用”状态。其实在你的产品推出之前，别人是没有工具可以用的，这种情况，叫做“0 分”。你的产品只要基本的将数据变成曲线就可以了，已经能做到 60 分了，再加上基本的导航，80 分就到了。当然，做出科幻片似的前端，确实能做到 100 分。但我们要这么想，从 0 分到 80 分，用户的快感，和 80 分到 100 分的快感，哪个更强呢？从 80 分做到 100 分，需要花多大的精力呢？对于软件开发本身来说，正确的方式就是发布简单的版本，根据用户的反馈，进行一次一次的迭代，最后收敛到一个用户满意的结果。

不要把半成品推出去给别人使用

看到这句话，大家是不是觉得更前面说的“先可用，再好用”是矛盾的，瞬间有了想打我的冲动。其实这两个观点并不矛盾。我对于一个“产品”的观点是“先可用，再好用”，这里的“可用”不是说东西还没法稳定的运行就推出去，而是把最基本的功能跑通，满足用户最痛点的需求，然后再慢慢完善它。用户对于产品的自信，是不可恢复的。当用户认为一个产品垃圾之后，这个产品想要在用户心里变成非垃圾，是非常困难的。所以说，在推出一个产品的时候，一定是要一个“能用”的版本，千万不能把半成品推出去给别人用。

文档很重要

一般越牛的程序员越不愿意写文档。而另一方面，程序员觉得自己的产品易用性已经到达极限了，如果不会用，一定是用户自己智商的问题。简单来说，就是易用性已经到达了不需要文档的地步了。首先，这种令人发指的易用性是不存在的，其次，不同用户的知识面不同，工作领域也不通，不会用产品是非常正常的。一个好的文档，能让用户了解这个产品的背景，学习如何使用这个产品。这是推销产品的一个好方式。

除非只是一个脚本，否则一个易用的前端非常的重要

一个好用的前端，会使得产品的易用性是呈指数级增长，并且可以将用户拓展到非程序员。而且，想象一下，一个是对着黑洞洞的终端敲命令，一个是在 Chrome 中鼠标点点就完成工作，哪个更好用呢？有人说前端太难了，没法做。这里我要澄清下：前端不难。1 周入门，2 周开发个简单的 Demo 绰绰有余。比如 Python 的 bottle.py、web.py，Ruby 的 ruby on rails，再高级点的可以用 Angular JS。现在前端工程师非常难招，学习一下，以后工资能高点，这是个两全其美的方法。

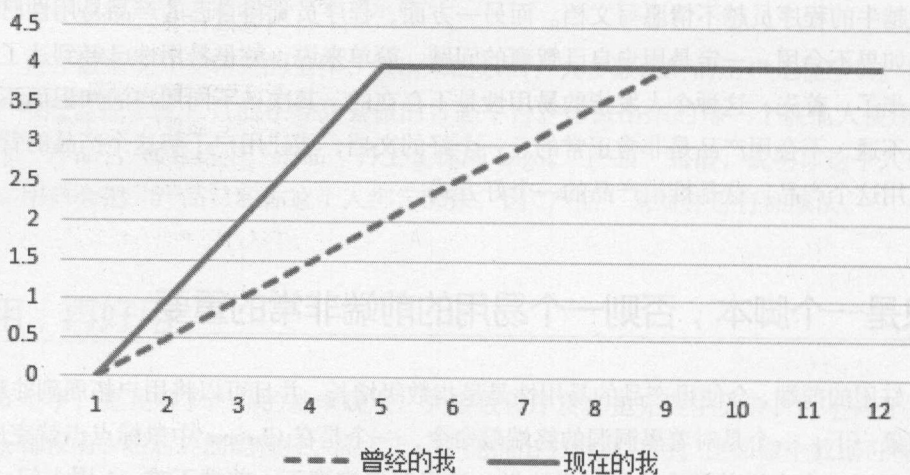
程序员这个种族

程序员的自信心基本都是爆表状态，也都是乐观主义者、完美主义者。乐观主义者体现在对于一个任务的完成时间，几乎所以程序员都会高估自己的战斗力，所以肯定是任务的实际完

成时间晚于程序员自己估计的时间；完美主义者体现在程序员认为自己能解决所有的问题，所以程序员对于自己推出的产品的要求就是完美无瑕。

程序员另一个特征就是容易同行相轻，这个想法是诸超提出来的，自己想想真是非常的正确。在我年少轻狂的时候，每当公司来了个新程序员，我第一反应是这个人技术肯定不如我，对于他问我的问题我也很反感，反正就是怎么看怎么不顺眼。在接触了解以后，对这位新程序员的评价，才能回到客观中。如下图，横坐标轴表示时间，纵坐标轴是新程序员的水平。假设这个新程序员的水平是4分，那么以前从我认为这人水平是0，到正确判断出这人水平是4，用了9个时间单位。在我意识到我也有这个“程序员互相看不起”的问题之后，现在改进到5个时间单位就能正确评价新程序员了。

我眼中的程序员



现在回头想想，如果大家也意识到自己有这个问题，就慢慢改吧，哈哈。

姚仁捷

2014年8月于上海



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview®
WWW.BROADVIEW.COM.CN

博文视点 · IT 出版旗舰品牌

博文视点诚邀精锐作者加盟

《C++Primer (中文版) (第5版)》、《淘宝技术这十年》、《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、咎辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

十年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金之计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于IT专业出版之巅。

以书为证彰显卓越品质

英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

· 专业的作者服务 ·

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

善待作者——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

尊重作者——我们尊重每一位作者的技术能力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

提升作者——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



联系我们

博文视点官网: <http://www.broadview.com.cn>

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

投稿电话: 010-51260888 88254368

投稿邮箱: jsj@phei.com.cn



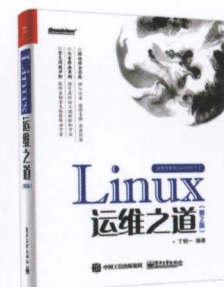
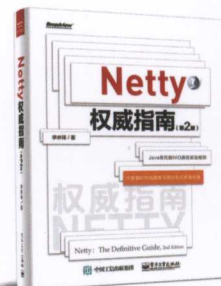
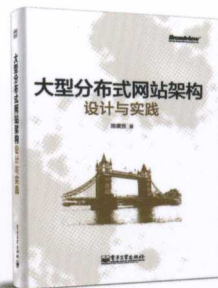
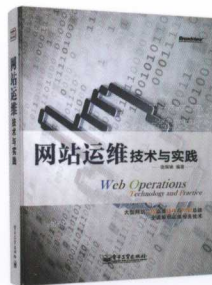
@博文视点Broadview



微信公众号 博文视点Broadview



十载耕耘奠定专业地位



欢迎反馈意见或投稿
邮箱: dongying@phei.com.cn
电话: 010-88254047
微信号: yingzidd

业内力荐

Zabbix是目前很流行的分布式图形化开源监控系统解决方案。它有健全灵活的监控数据采集、存储、告警规则配置以及图形化展示界面,已经被越来越多的互联网公司所应用,成为运维基础架构系统的重要组成部分。

三年前,姚仁捷作为聚力传媒(PPTV)的系统工程师,参与并承担PPTV生产环境运维监控系统的建设工作。在短短三个月的时间里,完成了方案选取、部署测试、小范围应用、全面上线的一整套工作,从无到有建设了覆盖数千台设备的运维监控系统。我作为项目负责人,对于他的成就感到非常欣慰。

本人作为姚仁捷的同事、朋友,向有志于运维自动化的朋友们郑重推荐此书!

陈文春
新浪运维中心总经理

Zabbix是一个用于基础架构监控和告警的开源解决方案,可以说是近几年来国内外中小型互联网企业中最流行的一种。它结合了Cacti绘图和Nagios的告警机制,并拥有非常简易的Web配置界面,再配合一些标准的监控模板,就能很轻松地上手了。姚仁捷曾经在PPTV负责过Zabbix大规模集群的工作,积累了丰富的经验和技巧,本书是他多年实践工作的结晶,从功能到部署,从原理到案例,全面讲解了Zabbix相关知识,值得一读!

程国强
携程网站运营中心系统研发高级总监

Zabbix是近几年涌现出来的开源企业级监控工具,集数据采集、图表绘制、报警等功能于一身,适合中小型企业快速地从无到有建立一个完备的监控体系。丰富的功能背后是极高的复杂性,特别在国内缺乏相关的中文资料的背景下,系统管理人员很难参透工具内在的精髓和局限,从而高效地将其用于大规模监控场景。姚仁捷是国内Zabbix大规模应用场景的实践先驱,在源代码级对Zabbix的工作机制进行过研究及优化。相信本书可以帮助后来者少走弯路,趋利避害,让Zabbix成为监控项目实施中的利器。

吴晓刚
携程网站运营中心系统研发总监

Zabbix是近年来非常流行的分布式监控工具,但是上手容易,精通难。八牛曾经是国内最大规模Zabbix集群的维护者,在Zabbix运维、优化及二次开发方面都有丰富的经验。本书作为他多年经验的结晶,不但详细阐述了Zabbix的部署运用,还有针对性地解析了Zabbix架构设计的关键点和具体实现方式,更列举了常见的疑难问题和解决方案,堪称监控领域难得的精品书籍,相信不同层次的读者都会有所收获。

饶琛琳
日志易总监



博文视点Broadview



@博文视点Broadview

上架建议: 计算机/监控运维

ISBN 978-7-121-29608-6



9 787121 296086 >

定价: 79.00元



责任编辑: 董英
封面设计: 吴海燕